o'reillys basics

2. Auflage

Praktischer Einstieg in MySQL mit PHP

- Behandelt MySQL 5.0 und 5.1 sowie PHP 5.2
- Schritt für Schritt zur datenbankgestützten Webanwendung
- Mit umfangreichem Beispielmaterial und vielen Software-Paketen auf CD-ROM



O'REILLY®

Sascha Kersken

ZWEITE AUFLAGE

Praktischer Einstieg in MySQL mit PHP

Sascha Kersken



Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

Kommentare und Fragen können Sie gerne an uns richten:
O'Reilly Verlag
Balthasarstr. 81
50670 Köln
Tel.: 0221/9731600
Fax: 0221/9731608
E-Mail: kommentar@oreilly.de

1. Auflage 2005
2. Auflage 2007

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte bibliografische Daten
sind im Internet über *http://dnb.ddb.de* abrufbar.

Lektorat: Inken Kiupel, Köln
Korrektorat: Sibylle Feldmann, Düsseldorf
Satz: Tim Mergemeier, reemers publishing services gmbh, Krefeld; www.reemers.de
Umschlaggestaltung: Michael Oreal, Köln
Produktion: Karin Driesen, Köln
Belichtung, Druck und buchbinderische Verarbeitung:
Druckerei Kösel, Krugzell; www.koeselbuch.de

ISBN-13 978-3-89721-717-1

Dieses Buch ist auf 100% chlorfrei gebleichtem Papier gedruckt.

Inhalt

Vorwort zur 2. Auflage

Das Letzte, was man findet, wenn man ein Werk schafft, ist die Erkenntnis, was man an seinen Anfang zu stellen hat.
Blaise Pascal

In einem offenen Brief wandten sich der Linux-Erfinder Linus Torvalds, der PHP-Schöpfer Rasmus Lerdorf und der MySQL-Begründer Michael »Monty« Widenius am 23.11.2004 gemeinsam an den Rat der Europäischen Union, um ihrem Protest gegen die Einführung von Softwarepatenten in der EU Ausdruck zu verleihen. Ausgerechnet dieser traurige Anlass zeigt, dass der Datenbankserver MySQL zu den wichtigsten Open Source-Projekten der Welt gehört. Zumindest handelt es sich um die mit Abstand verbreitetste Open Source-Datenbanksoftware. Die Kombination aus Apache-Webserver, PHP und MySQL ist zudem die beliebteste Plattform für Webanwendungen.

MySQL ist ein relationales Datenbankverwaltungssystem (RDBMS). Relationale Datenbanken bestehen aus miteinander verknüpften Tabellen, in denen Informationen so organisiert sind, dass sie sich sehr schnell heraussuchen, sortieren oder modifizieren lassen. Ihr besonderer Nutzen ergibt sich aus der Tatsache, dass es Schnittstellen für Programmiersprachen gibt, so dass Anwendungsprogramme auf die Datenbank zugreifen können.

Der wichtigste Anwendungsbereich von MySQL sind ohne Zweifel PHP-Webanwendungen. Da viele Webhoster diese Kombination bereits in relativ günstigen Tarifen anbieten, steht sie auch in der Praxis einem breiten Publikum zur Verfügung. Denn dass die Software selbst Open Source und damit unter anderem kostenlos ist, nützt bei Webanwendungen nichts, solange die Webhoster sie ihren Kunden nicht zur Verfügung stellen. Die meisten Websites liegen nicht auf eigenen Servern, sondern in gemietetem Webspace, wo die Software durch den Provider vorgegeben ist.

Darüber hinaus bringen die MySQL-Versionen 5.0 und 5.1 einige Neuerungen mit, die bisher vor allem in teuren kommerziellen Datenbanksystemen zu finden waren. Das macht MySQL zukünftig zu einer noch besseren Wahl für Geschäfts- oder Verwaltungsanwendungen aller Art.

Dieses Buch behandelt den Umgang mit dem Datenbankserver MySQL und seinen Bestandteilen sowie die Kombination mit der Programmiersprache PHP, um datenbankbasierte Webanwendungen zu schreiben. Zunächst erhalten Sie etwas Hintergrundwissen über Datenbanken und die Webprogrammierung, anschließend wird die Installation einer kompletten Serverumgebung für Webanwendungen beschrieben. Im weiteren Verlauf des Buchs erfahren Sie das Wichtigste über die Datenbankabfragesprache SQL und lernen am Rande einige Grundlagen der PHP-Programmierung kennen. Wenn Sie alle Kapitel durchgearbeitet haben, werden Sie in der Lage sein, mit dem MySQL-Datenbankserver umzugehen sowie MySQL-basierte PHP-Webanwendungen vom Datenbankentwurf bis zur Bereitstellung auf dem Webserver selbst zu schreiben.

Der vorliegende Band ist die zweite Auflage; er wurde an einige Neuerungen der nunmehr als stabil geltenden MySQL-Version 5.0 sowie der Betaversion 5.1 angepasst. Fortgeschrittene Datenbankfunktionen werden in dieser Ausgabe in einem eigenen Kapitel behandelt; die vorhandenen Ausführungen wurden leicht erweitert, und das neue Thema Trigger kam hinzu. Im Bereich der Webanwendungen werden nun auch JavaScript, DOM und Ajax behandelt. Schließlich wurden im Administrationskapitel die Themen Konfigurationsdateien und Replikation hinzugefügt.

Zielgruppe dieses Buchs

Um erfolgreich mit diesem Buch arbeiten zu können, sollten Sie einige Voraussetzungen erfüllen. Die wichtigste ist, dass Ihnen ein Computer mit einem der Linux-Betriebssysteme (oder einer anderen Unix-Variante) oder Windows zur Verfügung steht. Sie sollten mit Dateien und Verzeichnissen vertraut sein sowie die Arbeit mit mindestens einem Texteditor und natürlich mit einem Webbrowser beherrschen.

Ebenfalls erforderlich sind Grundkenntnisse in HTML, darauf wird in diesem Buch nicht eingegangen. PHP-Vorkenntnisse sind keine absolute Notwendigkeit, weil die benötigten Bestandteile der Sprache hier erläutert werden, aber den maximalen Nutzen dürfte dieses Buch einigermaßen erfahrenen PHP-Programmierern bieten, die ihre Kenntnisse – und ihre Websites – um eine Datenbank erweitern möchten.

Aufbau des Buchs

Dieses Buch umfasst neun Kapitel und vier Anhänge. In den Kapiteln werden folgende Themen behandelt:

- In Kapitel 1, *Einführung*, wird zunächst kurz auf die Entwicklungsgeschichte der Datenbanken eingegangen. Auch die grundlegende Funktionsweise von – insbesondere relationalen – Datenbanken sowie datenbankbasierten Webanwendungen wird angesprochen. Als Letztes werden die zentralen Konzepte und Fähigkeiten von MySQL selbst besprochen.
- Kapitel 2, *Installation und Inbetriebnahme*, beschreibt die Einrichtung und Basiskonfiguration eines kompletten LAMP- oder WAMP-Systems, das heißt Apache-Webserver, MySQL-Datenbank und PHP unter Linux beziehungsweise Windows.
- In Kapitel 3, *Die erste Webanwendung*, wird Schritt für Schritt die Entwicklung einer kleinen datenbankbasierten PHP-Anwendung von der Datenbankplanung bis zur Veröffentlichung im Webserver-Verzeichnis behandelt.
- Kapitel 4, *Mit MySQL arbeiten*, stellt zwei verschiedene Tools zum Bearbeiten von MySQL-Datenbanken vor: den mitgelieferten Kommandozeilenclient mysql zur manuellen Eingabe von Befehlen und den PHP-basierten Webclient phpMyAdmin.
- Kapitel 5, *Datenbanken entwerfen und erstellen*, ist gewissermaßen der erste Teil eines umfangreichen SQL-Tutorials: Zunächst werden Modelle und Vorgehensweisen für den Datenbankentwurf angesprochen, anschließend werden ausführlich alle wichtigen SQL-Anweisungen zur Datenbank- und Tabellenerstellung behandelt.
- Kapitel 6, *SQL-Abfragen*, ist der zweite, umfangreichere Teil der SQL-Einführung. In diesem Kapitel lernen Sie alle erforderlichen SQL-Anweisungen zum Auswählen und Bearbeiten von beliebig verknüpften Datenbanktabellen kennen.
- In Kapitel 7, *Fortgeschrittene Datenbankfunktionen*, werden einige Features vorgestellt, die in anderen Datenbanksystemen schon länger existieren, in MySQL aber erst in neueren Versionen eingeführt wurden.
- In Kapitel 8, *Webanwendungen mit PHP und MySQL*, werden einige umfangreichere datenbankbasierte PHP-Anwendungen vorgestellt. Zuvor erhalten Sie eine Einführung in die wichtigsten Konzepte von PHP selbst und einen systematischen Überblick über die MySQL-Datenbankschnittstellen der Sprache sowie über die Grundlagen von JavaScript, DOM und Ajax.
- Kapitel 9, *MySQL-Administration*, bietet eine Übersicht zu den wichtigsten Themen der Administration des MySQL-Servers selbst: Benutzer- und Passwortverwaltung, Optionen und Parameter der wichtigsten mit MySQL gelieferten (Hilfs-)Programme, Konfigurationsdateien, Datenimport und -export sowie Replikation.

In den Anhängen werden die folgenden Themen behandelt:

- Anhang A, die *Kurzreferenz*, bietet eine kurze Übersicht über die Syntax der wichtigsten SQL-Anweisungen sowie der PHP-Funktionen für den Datenbankzugriff zum schnellen Nachschlagen.
- In Anhang B, *Sonstige APIs*, werden im Schnellverfahren die MySQL-Schnittstellen der Programmiersprachen Perl, Java und Ruby sowie des Web-Frameworks Ruby on Rails vorgestellt.
- Anhang C, *Weitere Clients*, stellt einige andere Clientprogramme zum Arbeiten mit MySQL-Datenbanken vor, die Sie alternativ oder ergänzend zu *mysql* und *phpMyAdmin* einsetzen können.
- In Anhang D, *Ressourcen und Tools*, werden einige ergänzende Bücher und Websites zu den Themen dieses Buchs empfohlen.

Inhalt der beiliegenden CD-ROM

Damit Sie mit MySQL und PHP sofort in die Praxis einsteigen können, liegt diesem Buch eine CD-ROM mit der benötigten Software und allen Beispieldateien bei. Einen genauen Überblick über ihren Inhalt erhalten Sie, wenn Sie die im Wurzelverzeichnis enthaltene Datei *index.html* in einem Browser öffnen.

Im Einzelnen enthält die CD-ROM Folgendes:

- Die aktuellen Versionen (Stand: Mai 2007) der benötigten Software: Apache 2.2, PHP 5.2, MySQL 4.1, MySQL 5.0 und MySQL 5.1 sowie phpMyAdmin 2.10, jeweils für Linux und Windows.
- XAMPP – ein Server-Komplettpaket mit vorkonfigurierten Versionen der gesamten oben genannten Software, ebenfalls für Linux und Windows.
- Alle für die Anwendungen im Buch benötigten Datenbanken als SQL-Dateien zum Import in den MySQL-Server.
- Die im Buch entwickelten PHP-Anwendungen sowie weitere kommentierte PHP-Skripten, die die Praxisanwendung vervollständigen.
- Eine Linkliste zum Thema MySQL und PHP – die elektronische Version von Anhang D.

Beispiele

Die meisten Beispiele in diesem Buch stammen aus dem Umfeld eines (fiktiven) Online-Reisebüros, das Flugreisen in europäische Metropolen und die zugehörigen Hotelaufenthalte anbietet. Da es hier um die Technik zur Erstellung datenbankbasierter PHP-Anwendungen geht, wurden tief gehende Designaspekte absichtlich weggelassen.

Beachten Sie bitte auch, dass die Reisebüro-Datenbank nicht in allen Punkten realistisch ist. Aus Gründen der Übersichtlichkeit waren einige Vereinfachungen nötig. Beispielsweise wird die Anzahl freier und belegter Plätze für die Flugzeuge und Hotels weder gespeichert noch überprüft. Ein echtes Reisebüro greift für so etwas aber ohnehin meist auf das Buchungssystem eines externen Reiseveranstalters zu. Ein weiterer Abstrich gegenüber der Realität ist etwa der identische Preis für jedes Zimmer eines Hotels, und das unabhängig von der Saison.

Typografische Konventionen

In diesem Buch werden folgende typografische Konventionen verwendet:

Kursivschrift

Wird für Datei- und Verzeichnisnamen, E-Mail-Adressen und URLs, für Schaltflächenbeschriftungen, Menübefehle und -optionen sowie bei der Definition neuer Fachbegriffe und für Hervorhebungen verwendet.

`Nichtproportionalschrift`

Wird für Codebeispiele und Variablen, Funktionen, Befehlsoptionen, Parameter, Klassennamen und HTML-Tags verwendet.

`Nichtproportionalschrift fett`

Bezeichnet Benutzereingaben auf der Kommandozeile.

`Nichtproportionalschrift kursiv`

Kennzeichnet innerhalb von Codebeispielen Platzhalter, die Sie durch Ihre eigenen spezifischen Angaben ersetzen müssen.



Die Glühbirne kennzeichnet einen Tipp oder einen generellen Hinweis mit nützlichen Zusatzinformationen zum Thema.



Der Regenschirm kennzeichnet eine Warnung oder ein Thema, bei dem man Vorsicht walten lassen sollte.



In Kästen mit einem Mikroskop wird ein Thema genauer unter die Lupe genommen.

Danksagungen

Da dies eine Neuauflage ist, danke ich zunächst vor allem den Lesern der ersten Auflage. Ihr Interesse (und ihre Kaufentscheidung) hat die zweite Auflage erst möglich gemacht. Zudem erhielt ich hier und da Anregungen und Fehlermitteilungen per E-Mail, die geholfen haben, diese Auflage noch besser zu machen. Auch den Teilnehmern meiner regelmäßigen MySQL-Kurse im Linuxhotel (*http://www.linuxhotel.de*) bin ich sehr dankbar, denn sie haben nicht nur gern mit dem Buch gearbeitet, sondern auch fleißig Fehler gefunden und mich an manchen Stellen auf Unklarheiten und Ergänzungsbedarf hingewiesen.

Weiterer Dank gilt dem O'Reilly Verlag, insbesondere den Lektoren Michael Gerth (erste Auflage) und Inken Kiupel (zweite Auflage), die das Projekt mit Geduld und Kompetenz begleitet haben.

Dank gebührt auch den Entwicklern der in diesem Buch behandelten Open Source-Software – damit meine ich nicht nur den MySQL-Datenbankserver, sondern beispielsweise auch den Webserver Apache und die Programmiersprache PHP. Dass diese Menschen den Mut besitzen, der Öffentlichkeit das Ergebnis jahre- und nächtelanger Entwicklungsarbeit frei (und nicht nur kostenlos) zur Verfügung zu stellen, ist gewiss keine Selbstverständlichkeit und verdient Anerkennung.

Wenn auch Sie von Open Source-Software profitieren – wovon auszugehen ist, wenn Sie gerade dieses Buch lesen –, können Sie dieser Anerkennung auf vielfältige Weise Ausdruck verleihen. Beispielsweise können Sie Organisationen wie die Apache Software Foundation oder die Free Software Foundation mit Geldspenden unterstützen oder sich eine kommerzielle MySQL-Lizenz kaufen (die Ihnen das zusätzliche Recht verleiht, den MySQL-Server selbst in kommerzielle Software einzubetten).

Auch Beiträge zu den Softwareprojekten selbst sind immer willkommen. Sie brauchen nicht gleich eine umfangreiche Erweiterung zu programmieren, sondern helfen auch schon, wenn Sie gefundene Fehler in der Software oder ihrer Dokumentation an die Entwickler weitergeben – in Anhang D finden Sie die Adressen entsprechender Websites und Mailinglisten.

Darüber hinaus sollten auch Sie sich gegen Softwarepatente in der EU engagieren! Es gibt keine größere Gefahr, die der freien Software droht – nicht umsonst wird die umstrittene Patentrichtlinie vor allem von der Konkurrenz der Open Source-Projekte gefördert: einigen großen Unternehmen, die kommerzielle Software entwickeln und verkaufen. Gute Ausgangspunkte für konkrete Aktionen sind die Websites *http://www.nosoftwarepatents.com* und *http://www.ffii.de*. Dort erhalten Sie beispielsweise die Anschriften der verschiedenen EU-Abgeordneten sowie Informationen über Online- und Offline-Protestaktionen.

Zu guter Letzt danke ich von ganzem Herzen meiner Frau Tülay und meinem Sohn Leon, die auch für dieses Buch wieder viel zu oft und zu lange auf mich verzichten mussten.

In diesem Kapitel:

- Datenbanken
- Datenbankgestützte Anwendungen
- Grundlegendes zu MySQL

KAPITEL 1

Einführung

Wo nichts am rechten Platz liegt, da ist Unordnung.
Wo am rechten Platz nichts liegt, ist Ordnung.
Bertolt Brecht

In diesem Kapitel werden zunächst die historischen und theoretischen Hintergründe der Datenbanktechnik erläutert. Anschließend wird auf den Aufbau datenbankbasierter Anwendungen – und insbesondere Webanwendungen – eingegangen. Zum Schluss erhalten Sie die wichtigsten Informationen über das MySQL-Projekt selbst.

Datenbanken

Eine *Datenbank* (englisch *database*) ist eine computergestützte Sammlung gleichartiger Informationen, die sich auf optimierte Art und Weise durchsuchen, sortieren und manipulieren lässt. Der Begriff bezeichnet genau genommen nur die gespeicherten Daten selbst. Mittlerweile hat es sich aber eingebürgert, dass auch die Software zur Verwaltung dieser Daten, das Datenbankverwaltungssystem (*DataBase Management System*, kurz DBMS), einfach als Datenbank bezeichnet wird.

Motivation der Datenbankentwicklung

Eine der größten Herausforderungen moderner Computeranwendungen ist die rationelle Speicherung und Verwaltung der benötigten Daten. Selbstverständlich könnten alle Informationen in einzelnen Dateien beliebiger Formate abgelegt werden; jede Programmiersprache enthält Funktionen zum Öffnen, Lesen, Schreiben und sogar Durchsuchen von Dateien auf einem Datenträger. Allerdings bringt der Einsatz normaler Dateien eine Reihe gravierender Nachteile mit sich:

- Die in einfachen Dateien gespeicherten Informationen stehen zunächst in keinerlei Bezug zueinander. Schwierigkeiten bei der Suche und Inkonsistenzen durch die mehrfache Speicherung bestimmter Daten sind die Folge.

- Das ständige Öffnen, Ändern und Schließen vieler Dateien benötigt ziemlich viel Rechenzeit und belastet den Arbeitsspeicher. Zwar könnte man die benötigten Daten auch beim Programmstart in den Arbeitsspeicher laden (soweit sie hineinpassen) und dort bearbeiten, aber auf diese Weise käme es bei einem eventuellen Programmabsturz oder Stromausfall zu Datenverlusten.
- Sobald mehrere Anwendungen gleichzeitig dieselben Daten ändern möchten, kommt es – gelinde gesagt – zu Ungereimtheiten. Stellen Sie sich beispielsweise vor, Anwendung 1 speichert eine Änderung an einer Datei, an der Anwendung 2 noch arbeitet. Wenn Anwendung 2 später ihre eigenen Änderungen speichert, verschwinden diejenigen von Anwendung 1 im Daten-Nirvana.
- Verschiedene Betriebssysteme erlauben unterschiedliche Formate, Verzeichnisstrukturen und Dateinamen. Dies erschwert die Programmierung kompatibler Anwendungen.
- Immer mehr Programme arbeiten verteilt über lokale Netzwerke oder das Internet. Der gemeinsame Zugriff auf Dateien und Verzeichnisse über ein Netzwerk führt zu neuen Herausforderungen und Kompatibilitätsproblemen.

All diese Schwierigkeiten lassen sich jedoch effizient vermeiden, wenn Sie zur Datenverwaltung und -bearbeitung ein Datenbanksystem einsetzen. Das ist nicht weiter verwunderlich, denn genau solche Überlegungen haben seit etwa 1965 zur Entwicklung der ersten Datenbanken geführt.

Kurze Geschichte der Datenbanken

Die Idee, Informationen über mehrere gleichartige Elemente geordnet abzulegen, entstand schon lange vor der Computertechnik. Denken Sie zum Beispiel an hölzerne Karteikästen, Aktenordner oder Terminkalender. Im Zuge der fortschreitenden Verlagerung von Bürotätigkeiten in die EDV lag es nahe, entsprechende Organisationsformen für Daten im Computer nachzubilden.

Allerdings gab es zu Beginn der Computergeschichte, bis in die 60er-Jahre hinein, ein technisches Problem: Die Daten wurden zur dauerhaften Aufbewahrung in Lochkarten oder Lochstreifen gestanzt. Diese konnten ausschließlich sequenziell, das heißt der Reihe nach, eingelesen und verarbeitet werden. Eine solche Form der Informationsspeicherung erlaubt also keine Zugriffe in beliebiger Reihenfolge (engl. *random access*) und somit keine gezielte Suche nach bestimmten Merkmalen. Für den Betrieb datenbankähnlicher Strukturen sind derartige mechanische Speichermedien deshalb ungeeignet.

Erst die Einführung der magnetischen Datenträger lieferte die notwendige Grundvoraussetzung für den Datenbankbetrieb. Zunächst wurden Magnetbänder verwendet, die ihrer Bauart gemäß ebenso sequenziell waren wie die Lochstreifen oder -kartenstapel. Allerdings konnte geeignete Software für den benötigten Random-Access-Betrieb sorgen. Den endgültigen Durchbruch brachten aber erst die seit 1973 entwickelten Festplatten.

Die ersten Datenbanksysteme waren hierarchisch organisiert und ähnelten so eher den heutigen Dateisystemen mit ihrer verschachtelten Verzeichnisstruktur. 1970 entwarf *Edgar F. Codd* bei IBM das Modell der *relationalen Datenbank*, das MySQL und den meisten anderen Datenbanksystemen noch heute weitgehend zugrunde liegt. Ein wesentlicher Bestandteil von Codds Entwurf war eine auf englischer Umgangssprache basierende Abfragesprache namens SEQUEL (*Structured English Query Language*), die später aus rechtlichen Gründen in SQL (*Structured Query Language*) umbenannt wurde.

IBM selbst implementierte zunächst ein SQL-kompatibles relationales Datenbanksystem namens *System R*. Nach und nach entstanden zahlreiche weitere DBMS, die ebenfalls dem relationalen Modell genügten (*Relational Database Management Systems* oder *RDBMS*). Dazu gehörten auf der einen Seite High-End-Systeme wie Oracle, Informix oder IBM DB2, auf der anderen Seite aber auch Einzelplatz- oder Desktop-Datenbanken wie dBase, Microsoft Access oder FileMaker. Die heute dominierenden Systeme sind die Open Source-Projekte MySQL und PostgreSQL sowie die kommerziellen Produkte Microsoft SQL Server, Oracle und DB2.

Seit den 80er-Jahren wurden auch verschiedene nicht relationale Datenbanksysteme entwickelt. Dazu gehören insbesondere folgende Arten:

Objektorientierte Datenbanken
: Die Daten werden in einer komplexen Objekthierarchie gespeichert. Dies erlaubt flexiblere Verknüpfungen als bei den relationalen Tabellen.

Objektrelationale Datenbanken
: Einige Datenbanken mischen relationale und objektorientierte Ansätze auf unterschiedliche Weise.

XML-Datenbanken
: Da das XML-Format in vielen Bereichen der Computertechnik sehr wichtig geworden ist, liegt es nahe, auch Datenbanken im XML-Format oder zur Speicherung von XML-Strukturen einzusetzen.

Diese moderneren Ansätze konnten sich bisher nicht gegen die Übermacht der relationalen Datenbanken behaupten, und es sieht auch nicht danach aus, dass sich dies in nächster Zeit ändern würde. Die Standardisierung der (inzwischen ISO-genormten) Abfragesprache SQL, die Verfügbarkeit von leistungsfähigen Tools und Schnittstellen sowie eine breite Basis erfahrener Datenbankadministratoren machen die relationalen Systeme auch weiterhin zur ersten Wahl.

Relationale Datenbanken

In einem relationalen Datenbankverwaltungssystem (RDBMS) werden alle Daten in *Tabellen* abgelegt. Der Name stammt daher, dass die Tabellen aus mathematischer Sicht *Relationen* sind. Jede Zeile einer Tabelle wird als *Datensatz* (englisch *Record*)

bezeichnet. Sie enthält in den einzelnen Spalten verschiedene Informationen über einen einzelnen Gegenstand, der in diesem Zusammenhang *Entity* (zu Deutsch *Entität*) genannt wird. In jeder Tabellenspalte befindet sich eine bestimmte Informationskategorie. In der Datenbanktheorie heißt eine solche Kategorie *Attribut* eines Datensatzes, in der Praxis wird sie auch oft als *Datenfeld* bezeichnet.

Das Interessanteste an relationalen Datenbanken ist die Möglichkeit, die Informationen in verschiedenen Tabellen miteinander zu verknüpfen. Zu diesem Zweck wird jedem Datensatz einer bestimmten Tabelle ein spezielles Attribut mit eindeutigem Wert zugewiesen, der sogenannte *Primärschlüssel* (englisch *Primary Key*). Soll nun in einer anderen Tabelle auf einen solchen Datensatz Bezug genommen werden, braucht nur noch dieser Schlüssel aufzutauchen. So lässt sich die doppelte Speicherung von Daten (Redundanz) vermeiden, die durch Eingabefehler oder durch das Vergessen von Aktualisierungen sonst leicht zu Inkonsistenzen führen kann.

Ein einfaches Beispiel macht diese theoretischen Erläuterungen verständlicher. Stellen Sie sich vor, diverse Kunden buchen verschiedene Pauschalreisen. Tabelle 1-1 zeigt den (untauglichen) Versuch, die Details der Kunden und ihrer Reisen in einer gemeinsamen Tabelle abzulegen.

Tabelle 1-1: Nicht relationaler Tabellenentwurf

Name	Vorname	Wohnort	Reisedatum	Reiseziel	Verkehrsmittel
Schmitz	Klaus	Köln	02.06.2007	Paris	Reisebus
Müller	Andrea	Mannheim	07.06.2007	London	Bahn
Huber	Ludwig	München	02.06.2007	Paris	Reisebus
Jansen	Jutta	Hamburg	10.06.2007	Rom	Flugzeug
Becker	Peter	Bonn	07.06.2007	London	Bahn
Schmitz	Klaus	Köln	05.10.2007	Istanbul	Flugzeug

Die Probleme sind offenkundig: Mehrere Personen buchen gleiche Reisen, und umgekehrt bucht eine Person zwei verschiedene Reisen. Stellen Sie sich nun zum Beispiel vor, die Reise nach Paris wird auf den 03.06. verlegt oder wird mit der Bahn statt mit dem Bus durchgeführt. Dann müsste die entsprechende Änderung in allen Zeilen durchgeführt werden, in denen eine Paris-Buchung steht. Ein ähnlicher Fall träte ein, falls der Kunde Klaus Schmitz umziehen würde: In jeder seiner beiden Buchungen müsste der Wohnort separat geändert werden.

Das vorliegende Beispiel ist noch halbwegs übersichtlich. In der Praxis enthalten Tabellen aber oft Dutzende von Attributen und einige Hundert bis mehrere Millionen Datensätze. Einem Menschen wäre es unmöglich, solche unnötigen Änderungen an mehreren Stellen durchzuführen, und selbst für ein Computerprogramm wäre ein solches Vorgehen äußerst unökonomisch.

Die Lösung im Sinne des relationalen Modells besteht darin, die Informationen auf zwei Tabellen zu verteilen: Die Daten über die Kunden gehören in eine andere Tabelle als die Informationen über die Reisenden. Wie oben bereits erwähnt, wird jeder Datensatz durch ein eindeutiges Attribut (den Primärschlüssel) gekennzeichnet, um Verknüpfungen möglich zu machen. Da weder Reisende noch Pauschalreisen über ein »natürliches« Schlüsselmerkmal verfügen (wie zum Beispiel die ISBN bei Büchern oder das amtliche Kennzeichen bei Autos), wurde einfach eine Nummerierung hinzugefügt. In den Tabellen 1-2 und 1-3 sehen Sie die beiden entstandenen Datenbanktabellen.

Tabelle 1-2: Tabelle »Kunden«

KundeNr	Name	Vorname	Wohnort
K001	Schmitz	Klaus	Köln
K002	Müller	Andrea	Mannheim
K003	Huber	Ludwig	München
K004	Jansen	Jutta	Hamburg
K005	Becker	Peter	Bonn

Tabelle 1-3: Tabelle »Reisen«

ReiseNr	Reisedatum	Reiseziel	Verkehrsmittel
R001	02.06.2007	Paris	Reisebus
R002	07.06.2007	London	Bahn
R003	10.06.2007	Rom	Flugzeug
R004	05.10.2007	Istanbul	Flugzeug

Als Letztes muss nun die Verknüpfung zwischen den Kunden und ihren Reisen erfolgen. Wie bereits festgestellt wurde, können mehrere Kunden ein und dieselbe Reise buchen, aber genauso gut kann ein Kunde mehrere Reisen wahrnehmen (sofern sie nicht gleichzeitig stattfinden). Im Jargon der relationalen Datenbanken ist dies eine m:n-Relation – mehrere Datensätze des einen Typs können mit jeweils mehreren eines anderen Typs verknüpft werden. Die Reisenummern können also nicht als Verweis in der Kunden-Tabelle stehen, und umgekehrt geht es auch nicht. Für m:n-Relationen wird immer eine dritte Tabelle benötigt, die beliebig viele Kunden mit beliebig vielen Reisen verknüpfen kann. Tabelle 1-4 zeigt die Lösung, die alle Beziehungen aus der ursprünglichen Tabelle in Bezüge auf die Tabellen 1-2 und 1-3 umwandelt.

Tabelle 1-4: Die Datenbanktabelle »Buchungen«

BuchungNr	Kunde	Reise
B0001	K001	R001
B0002	K002	R002
B0003	K003	R001

Tabelle 1-4: Die Datenbanktabelle »Buchungen« (Fortsetzung)

BuchungNr	Kunde	Reise
B0004	K004	R003
B0005	K005	R002
B0006	K001	R004

In Abbildung 1-1 werden die Beziehungen zwischen den drei Tabellen noch einmal verdeutlicht. Eine genaue Erläuterung derartiger Relationsdiagramme finden Sie in Kapitel 5, *Datenbanken entwerfen und erstellen*.

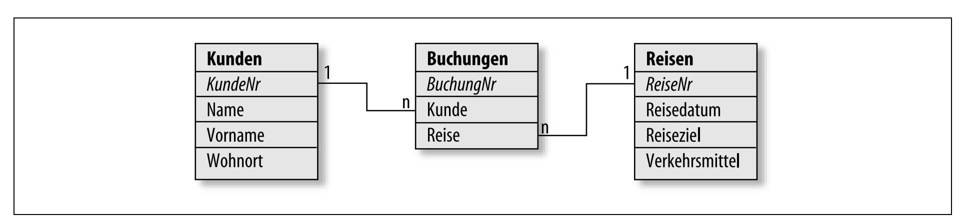


Abbildung 1-1: Grafische Darstellung der Relationen zwischen den drei Tabellen

Datenbankgestützte Anwendungen

In diesem Buch geht es um Webanwendungen, die ihren Datenvorrat aus einer MySQL-Datenbank schöpfen und neu hinzugekommene Informationen wieder in diese hineinschreiben. Das Prinzip einer Webanwendung unterscheidet sich ein wenig von einem allgemeinen Computerprogramm. In diesem Abschnitt erhalten Sie eine kurze Übersicht über datenbankgestützte Anwendungen im Allgemeinen und datenbankgestützte Webanwendungen im Besonderen.

Bei einem gewöhnlichen Anwendungsprogramm, das seine Informationen in einer Datenbank aufbewahrt, sind zwei Komponenten beteiligt: das Programm selbst sowie der Datenbankserver. »Server« ist hier gar nicht unbedingt im Sinne eines Netzwerkservers gemeint, sondern allgemeiner als Programm, das den Datenbankdienst zur Verfügung stellt. Damit Sie eine solche Lösung einsetzen können, muss eine Kommunikationsschnittstelle zwischen dem Programm und dem Datenbankserver bestehen.

Generell gibt es vier grundlegende Arten solcher Schnittstellen:

- Die Schnittstelle verbindet ein bestimmtes DBMS mit einer bestimmten Programmiersprache. Beispielsweise enthält PHP zwei verschiedene spezielle Schnittstellen für den Zugriff auf MySQL-Datenbanken.[1]

1 Die aktuelle Version PHP 5 bemüht sich, die einst besonders starke Verbindung zu MySQL zu relativieren und die MySQL-Schnittstelle als »eine von vielen« Datenbankschnittstellen in PHP zu behandeln.

- Die Schnittstelle wird mit der Datenbank geliefert und kann durch verschiedene Programmiersprachen angesprochen werden. MySQL enthält ab Werk diverse Bibliotheken, etwa für C oder Perl.
- Die Schnittstelle ist in die Programmiersprache eingebaut und kann auf unterschiedliche Datenbanken zugreifen. Beispiele dieses Typs sind die neuen, in diesem Buch ausführlich behandelten PHP Data Objects (PDO), Perl DBI und JDBC für Java (siehe Anhang B). Beachten Sie, dass meist ein zusätzlicher Treiber für konkrete DBMS erforderlich ist.
- Es handelt sich um eine allgemeine Drittanbieterschnittstelle, die verschiedene Datenbanken mit unterschiedlichen Programmiersprachen verknüpfen kann; auch hier wird unter Umständen ein konkreter Treiber benötigt. Die bekannteste Schnittstelle dieses Typs, Microsoft ODBC, ist seit vielen Jahren Bestandteil aller Windows-Versionen.

All diese verschiedenen Arten von Schnittstellen unterscheiden sich vor allem in der Art und Weise voneinander, wie die Anwendung eine Verbindung zum Datenbankserver herstellt. Auch die genauen Bezeichnungen der Befehle für den Datenbankzugriff können je nach Sprache und Schnittstelle unterschiedlich sein. Dennoch überwiegen die Gemeinsamkeiten: Ist die Verbindung erst einmal hergestellt, geht es darum, der Datenbank SQL-Abfragen zu übermitteln, ihre Antworten entgegenzunehmen und diese zu verarbeiten. Trotz geringfügiger Unterschiede in der SQL-Syntax und -Vollständigkeit der verschiedenen Datenbanksysteme funktioniert dieser Teil einer datenbankbasierten Anwendung fast immer gleich.



Neben den hier angesprochenen SQL-basierten Datenbankschnittstellen etablieren sich allmählich verschiedene Arten der Abstraktion. Während PDO nur den Verbindungsaufbau verallgemeinert und danach wie üblich SQL-Strings an die Datenbank sendet, abstrahieren einige neuere Lösungen auch die Kommunikation zwischen Programmiersprache und Datenbank durch standardisierte Funktionsaufrufe.

Noch einen Schritt weiter gehen sogenannte Objekt-relationale Mapper (ORM), die eine Datenbankstruktur automatisch einer Klassen- und Objekthierarchie zuordnen. Zwei bekannte Beispiele sind Active Record aus dem Web-Framework Ruby on Rails sowie Microsoft ADO.NET. Ersteres wird im Zusammenhang mit MySQL kurz in Anhang B vorgestellt.

Abbildung 1-2 zeigt das Funktionsprinzip einer datenbankgestützten Anwendung. Da zwei Programme beteiligt sind, wird diese Konstellation als *Two-Tier-Lösung*[2] bezeichnet. Eine solche Anwendung würde beispielsweise auf einem PC in den Geschäftsräumen des Reisebüros laufen. Es besteht eine direkte Verbindung zwi-

2 Das englische Wort »tier« bedeutet Schicht oder Lage.

schen dem für den Benutzer sichtbaren Anwendungsprogramm und dem Datenbankserver. Die Benutzer interagieren nur mit dem Anwendungsprogramm; von der Datenbank, auf die im Hintergrund zugegriffen wird, bekommen sie nichts mit.

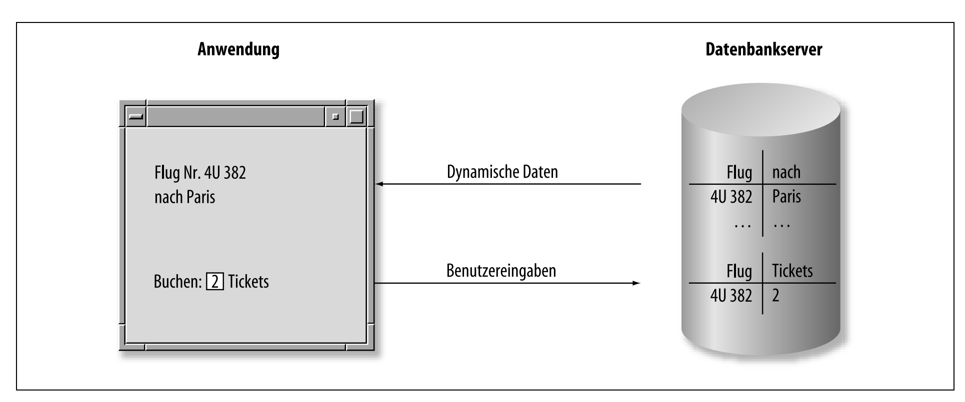


Abbildung 1-2: Two-Tier-Anwendung mit Datenbankserver und Benutzeroberfläche

Das Beispiel in der Abbildung besteht aus zwei Arbeitsschritten: Zunächst werden Daten aus einer Tabelle der Datenbank gelesen, hier konkret Nummer und Ziel eines Flugs. Diese Informationen stellt das Anwendungsprogramm auf dem Bildschirm dar. Anschließend kann der Benutzer – hier wahrscheinlich ein Angestellter des Reisebüros – die Anzahl der Tickets eingeben, die für diesen Flug reserviert werden sollen. Diese Eingabe wird über die Schnittstelle wieder an den Datenbankserver geschickt, der sie in einer anderen Tabelle ablegt.

Technisch gesehen, sendet die Anwendung zunächst eine SQL-Abfrage an den Datenbankserver, um die Daten auszulesen. Ohne auf Details einzugehen, die in späteren Kapiteln vertieft werden: Bei dieser ersten Kommunikation mit dem RDBMS handelt es sich um eine sogenannte *Auswahlabfrage*. Sie liefert Datensätze aus der Datenbank, die bestimmten Kriterien entsprechen. Auch die Benutzereingabe wird wieder in die Datenbank geschrieben, was ebenfalls durch eine SQL-Abfrage bewerkstelligt wird. Dies erledigt entweder eine *Änderungsabfrage* oder eine *Einfügeabfrage* – je nach konkreter Organisation der Datenbank muss für die Flugbuchung ein Datensatz modifiziert oder hinzugefügt werden.

Eine datenbankgestützte *Webanwendung* ist etwas komplizierter. Das »Problem« besteht darin, dass die Benutzer von internetbasierten Anwendungen in der Regel kein spezialisiertes Anwendungsprogramm besitzen, um auf die Dienste eines bestimmten Anbieters zuzugreifen, sondern lediglich einen allgemeinen Webbrowser. Dieser ist weder in der Lage, mit dem Datenbankserver zu kommunizieren, noch kann er selbst spezialisierte Bearbeitungsschritte bereitstellen. Er kann lediglich durch HTML formatierte Textinformationen anzeigen. Diese HTML-Dokumente können Formulare mit Eingabefeldern und anderen Auswahlelementen enthalten; die Benutzereingaben werden auf Knopfdruck an den Server zurückgesendet.

Aus diesen Gründen ist jede datenbankbasierte Webanwendung eine *Three-Tier-Lösung*: Beteiligt sind der Datenbankserver, der Webserver und der Browser des Benutzers.[3]

Die eigentliche Anwendung wird hier auf dem Webserver ausgeführt. Sie liest – wieder mithilfe einer SQL-Auswahlabfrage – die Informationen über den Flug aus der Datenbank. Sie erstellt aus diesen Daten und einigen festgelegten Textbausteinen wie »Flug Nr.« oder »Buchen« ein HTML-Dokument, das an den Browser des Benutzers übermittelt wird. Der Browser zeigt das Dokument gemäß den enthaltenen HTML-Steueranweisungen an. Das Dokument enthält ein Formular mit einem Textfeld zur Eingabe der gewünschten Ticketanzahl sowie einen Absende-Button. Sobald der Benutzer diesen betätigt, wird die eingegebene Anzahl an den Server zurückgeschickt. Ein weiteres Webserver-Anwendungsprogramm nimmt sie entgegen und schreibt sie per Änderungs- oder Einfügeabfrage in die Datenbank. Abbildung 1-3 illustriert den beschriebenen Ablauf.

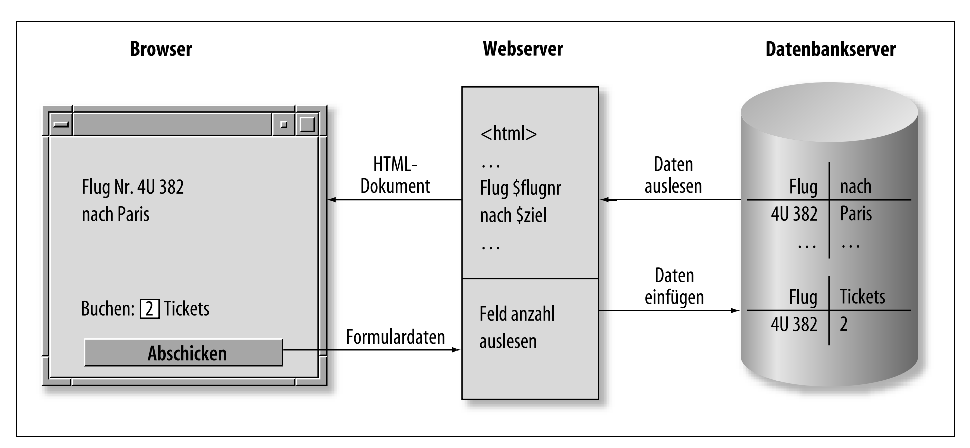


Abbildung 1-3: Three-Tier-Anwendung mit Datenbankserver, Webserver und Browser

Diese Variante der Anwendung ist eher für Endkunden geeignet, die über das Internet eine Reise buchen möchten. Allerdings ist auch innerhalb geschlossener Organisationen in den letzten Jahren der Trend zu beobachten, dass browserbasierte Webanwendungen die spezialisierten Einzelanwendungsprogramme ablösen. Sie werden im Gegensatz zum externen Internet als Intranet-Anwendungen bezeichnet. In einer gemischten IT-Landschaft mit verschiedensten Computer- und Betriebssystemen ist diese Methode einfach praktischer als die ständige Anpassung plattformabhängiger Programme.

3 Altmodische Webanwendungen werden von Interpretern außerhalb des Webservers ausgeführt (sogenannte CGI-Programme); dies führt letztlich sogar zu einer »Four-Tier-Anwendung« (wenngleich der Begriff nicht gebräuchlich ist).

Das schwierigste Problem einer Webanwendung geht aus dem Beispiel und der Abbildung übrigens nicht hervor. Webserver und -browser kommunizieren über das Protokoll HTTP (Näheres siehe Kapitel 3, *Die erste Webanwendung*), das eine grundsätzliche Schwäche aufweist: Es ist zustandslos (englisch *stateless*), das heißt, jede Anfrage des Browsers mit anschließender Antwort des Servers ist ein einzelner, in sich geschlossener Vorgang. Oftmals müssen aber längere Transaktionen über das Web stattfinden, zum Beispiel bei Bestell-Sites mit Warenkorbsystem. Hier muss ein Weg gefunden werden, den nicht vorhandenen Zusammenhang zwischen den einzelnen Seitenabrufen anderweitig herzustellen. In Kapitel 8, *Webanwendungen mit PHP und MySQL*, lernen Sie zu diesem Zweck Sessions und Cookies kennen.

Eine interessante Neuentwicklung sind übrigens *Ajax-Anwendungen*. Sie kombinieren einige bekannte Techniken – JavaScript, DOM und asynchrone HTTP-Anfragen – zu neuartigen Webanwendungen, die sich so flüssig bedienen lassen wie Desktop-Programme. Sie laden nämlich nicht mehr bei jeder Änderung die gesamte Webseite nach, sondern tauschen nur den jeweils relevanten Bereich aus. Das Konzept und einige Anwendungsbeispiele werden ebenfalls in Kapitel 8 vorgestellt.

Grundlegendes zu MySQL

Der Vorläufer der Datenbank MySQL wurde ab 1994 von dem Schweden Michael »Monty« Widenius für die Firma TcX entwickelt, basierend auf seinem Tool UNIREG von 1979. Ursprünglich ging es nur darum, schnellere Routinen für den Zugriff auf mSQL-Datenbanken zu schreiben; erst allmählich wurde ein eigenständiges DBMS-Projekt daraus.

1995 stieß David Axmark zu dem Projekt. Er setzte sich dafür ein, dass MySQL frei im Internet zur Verfügung gestellt wurde. Dies geschieht seit 1996 zunächst in Form von Binärdistributionen und bald auch als Open Source-Software unter der GNU General Public License (GPL), unter der zum Beispiel auch das Betriebssystem Linux verbreitet und weiterentwickelt wird.

Die MySQL-Entwickler gründeten die Firma MySQL AB (*http://www.mysql.com*), die sich um die Weiterentwicklung, Verbreitung und Dokumentation des Datenbanksystems kümmert. Ihr Geld verdient die Firma mit Support für größere Firmenkunden. Daneben wird eine kommerzielle MySQL-Lizenz angeboten; wer den MySQL-Server selbst in eine eigene kommerzielle Anwendung integrieren möchte, muss diese Lizenz erwerben.

MySQL erlangte sehr schnell eine immense Verbreitung. Der Schwerpunkt lag von Anfang an auf Webanwendungen. In diesem Bereich ist MySQL so populär, dass die verbreitetste Plattform für Webanwendungen als *LAMP* bezeichnet wird: Betriebssystem *Linux*, Webserver *Apache*, Datenbank *MySQL* und Programmiersprache *PHP* (manchmal auch Perl oder Python). Die entsprechende Windows-

Variante heißt *WAMP*; unter Windows 2000, XP oder Windows Server 2003 sind solche Systeme durchaus praxistauglich. Im nächsten Kapitel erfahren Sie Schritt für Schritt, wie Sie ein LAMP- beziehungsweise WAMP-System installieren und konfigurieren können.

Was für den Schwerpunkt Web- und Netzwerkanwendungen spricht, ist die Tatsache, dass MySQL von Anfang an ein Client-Server-System war: Der Datenbankserver kümmert sich um die eigentliche Datenbankverwaltung, während für die Steuerung durch den Benutzer verschiedene Clients (von der Kommandozeile über grafische Oberflächen bis hin zur Webanwendung) eingesetzt werden. Server und Client können sowohl auf demselben Rechner laufen als auch an beliebigen Punkten im Netzwerk oder Internet.

Neben der kostenlosen Verfügbarkeit war einer der großen Pluspunkte von MySQL stets die Geschwindigkeit; gerade durch das Weglassen verschiedener Features gelang es den Entwicklern, den Datenbankserver zu beschleunigen.

Dies impliziert gleichzeitig den größten Nachteil von MySQL: Wichtige Features, die eine »ausgewachsene« relationale Datenbank ausmachen, wurden lange Zeit nicht implementiert, zum Beispiel Unterabfragen oder die Transaktionsunterstützung. Für Letztere wurde inzwischen eine MySQL-typisch pragmatische Lösung gefunden: MySQL kann unterschiedliche Tabellentypen verarbeiten, von denen einige Transaktionen beherrschen. Sie können sich also für jede einzelne Datenbank aussuchen, ob Sie Transaktionen benötigen oder zugunsten einer besseren Performance darauf verzichten möchten.

Noch immer kann MySQL nicht ganz mit der ungeheuren Ausstattung manch anderer RDBMS wie etwa Oracle oder PostgreSQL mithalten. Auch der SQL99-Standard wird nicht vollständig unterstützt. Wie dieses Buch noch zeigen wird, können Sie mit MySQL *fast* alles machen – die wenigen Features, die noch nicht verfügbar sind, lassen sich oft durch einen (kleinen) zusätzlichen Aufwand in der Anwendungsprogrammierung kompensieren.



Wie die meisten Open Source-Anwendungen führt auch MySQL ein Tier im Logo, den Delfin *Sakila*. Die bekannten Tiere auf den O'Reilly-Covern regten ursprünglich wohl zu dieser Tradition an,[4] und spätestens seit dem Linux-Pinguin Tux kommt bessere Open Source-Software nicht mehr ohne Wappentier aus.

Bitte beachten Sie: Die Bezeichnung MySQL steht für ein ganzes Bündel von Software. Die wichtigste Komponente ist der MySQL-Server. Sein eigentlicher Programmname lautet *mysqld*, wobei das abschließende d für »Daemon« steht – in der

4 Einige neuere O'Reilly-Buchreihen verwenden keine Tiere. In der vorliegenden Basics-Reihe kommen beispielsweise Röntgenmotive von Alltagsgegenständen zum Einsatz.

Unix-Terminologie ein im Hintergrund laufendes Dienstprogramm. Hinzu kommt der Kommandozeilenclient mysql zur direkten Manipulation des Servers durch die Eingabe von SQL-Abfragen. Daneben werden noch zahlreiche Tools und Verwaltungsprogramme mitgeliefert.

Der MySQL-Server steht in drei verschiedenen Varianten zur Verfügung:

MySQL Community Server, Standard
: Dies ist die unter der GPL lizenzierte reine Open Source-Variante des MySQL-Servers.

MySQL Community Server, Max
: Auch diese Version ist Open Source-Software unter der GPL; sie enthält einige Erweiterungen gegenüber der Standardversion, zum Beispiel zusätzliche Tabellentypen. Wenn ein in diesem Buch behandeltes Feature nur in der Max-Version verfügbar ist, wird in der Regel darauf hingewiesen.

MySQL Enterprise
: Der Enterprise Server ist die kommerzielle Version des MySQL-Datenbankservers. Der Funktionsumfang ist in etwa identisch mit MySQL Max, aber zusätzlich gibt es direkten Support durch die MySQL AB sowie die bereits erwähnte Erlaubnis, den MySQL-Server in eigene kommerzielle Softwareprodukte einzubetten.



Neben diesen Varianten des MySQL-Servers bietet die MySQL AB noch einen Datenbankserver namens MaxDB an. Verwechseln Sie diesen nicht mit MySQL Max; es handelt sich vielmehr um die umbenannte SAP DB, die Datenbank des Warenwirtschaftssystems SAP, die vor einigen Jahren an die MySQL AB übergeben wurde und von dieser als Open Source-Datenbank weiterentwickelt wird.

Fähigkeiten von MySQL

Die folgende Liste enthält einen kurzen Überblick über die wichtigsten Funktionen des MySQL-Datenbankservers. Bei einigen von ihnen wird angegeben, seit welcher Version sie verfügbar sind – alle anderen wurden bereits in einer der frühesten Versionen eingeführt.

- Weitgehende Unterstützung von ANSI SQL 99
- Optimierte Binärdistributionen für Windows und zahlreiche Unix-Varianten wie Linux, MacOS X, Solaris oder FreeBSD
- Freie Wahl des Tabellentyps: geschwindigkeitsoptimierte MyISAM-Tabellen, transaktionsfähige InnoDB-Tabellen und weitere spezialisierte Typen
- Sehr detailliert einstellbare Benutzerrechte und Berechtigungen

- Eingebettete Datenbankbibliothek (*libmysqld*) für den Einbau der Serverkomponente in eigene Anwendungsprogramme. Bitte beachten Sie: Wenn Sie mit diesem Feature kommerzielle Software entwickeln, benötigen Sie eine kommerzielle MySQL-Lizenz.
- Replikation – automatische Spiegelung von Datenbanken auf mehrere Server (seit Version 4.0)
- Query-Caching – Zwischenspeicherung häufiger Abfragen (seit Version 4.0)
- Volltextsuche – schnelle Suche nach einzelnen Wörtern und Textbestandteilen (in Version 4.0 erweitert)
- Unterstützung für Unicode und frei wählbare Zeichensätze mit Konvertierung (seit Version 4.1)
- Unterabfragen (Subqueries) – das Ergebnis einer Abfrage kann als Einschränkungskriterium einer anderen dienen (in vollem Umfang seit Version 4.1)
- Räumliche und geometrische Daten gemäß GIS-Standard (seit Version 4.1)
- Stored Procedures – mehrzeilige SQL-Funktionen auf dem Datenbankserver selbst (seit Version 5.0)
- Trigger – SQL-Konstrukte, die bei jeder Abfrage eines bestimmten Typs automatisch aufgerufen werden; ideal etwa zur Überprüfung neu eingefügter Werte (seit Version 5.0)
- Multi-Master-Replikation – Replikation aus mehreren Quellen (seit Version 5.0)
- MySQL Cluster – automatische Zusammenarbeit von MySQL-Server-Instanzen auf mehreren Rechnern zur Performancesteigerung (seit Version 5.0)

MySQL 5.1 (zurzeit noch in der Betaphase) enthält vor allem folgende Neuerungen:

- Partitionierung – Speicherung von Tabellen über verschiedene Partitionen und physikalische Datenträger hinweg
- Zeilenbasierte Replikation – anders als bei der früheren MySQL-Replikation werden keine SQL-Anweisungen kopiert, sondern die Inhalte von Tabellenzeilen
- Event Scheduler – automatische Erledigung bestimmter Aufgaben zu festgelegten Zeiten (zum Beispiel Backups oder Anlegen neuer Logdateien)
- Logtabellen – einige Logs können nun nicht mehr nur in Dateien, sondern auch direkt in spezielle Tabellen der Verwaltungsdatenbank *mysql* geschrieben werden
- XML-Funktionen – einige spezielle SQL-Funktionen zur Verarbeitung von Textinhalten im XML-Format
- Plugin-API – standardisierte Schnittstelle zur Entwicklung eigener Erweiterungen des MySQL-Servers

Alpha, Beta, RC – Software-Entwicklungsstadien

Im Allgemeinen werden sowohl bei kommerzieller als auch bei Open Source-Software folgende Bezeichnungen für Entwicklungsstadien verwendet:

Alpha-Versionen
: Bieten eine reine Funktionsvorschau; sie sind meist instabil und kaum für den praktischen Einsatz geeignet.

Beta-Versionen
: Werden zum Testen unter realistischen Nutzungsbedingungen veröffentlicht; bei größeren Problemen kann sich aber noch einiges ändern.

Release Candidates (RC)
: Werden manchmal auch Gamma-Versionen genannt; es handelt sich um eine zur Veröffentlichung vorgesehene Version, an der nur noch ein paar letzte Fehler beseitigt werden.

Final
: Die als stabil geltende Version, die schließlich veröffentlicht wird.

In diesem Kapitel:

- Zu Unrecht gefürchtet: das Arbeiten mit der Konsole
- Apache 2 installieren und konfigurieren
- MySQL installieren
- PHP installieren
- phpMyAdmin einrichten

KAPITEL 2

Installation und Inbetriebnahme

Wer eine Hütte baut, fängt nicht mit dem Strohdach an.
Chinesisches Sprichwort

In diesem Kapitel erfahren Sie zunächst, wie Sie einen Computer als Server für MySQL-basierte PHP-Anwendungen einrichten können: Schritt für Schritt wird erläutert, wie Sie ein sogenanntes LAMP (Linux, Apache, MySQL, PHP) beziehungsweise WAMP-System (dasselbe unter Windows) installieren und in Betrieb nehmen.

Auf die Installation des jeweiligen Betriebssystems selbst wird hier nicht eingegangen. Alle Beschreibungen gehen davon aus, dass sich auf Ihrem Rechner bereits eine korrekt installierte, einigermaßen aktuelle Version von Windows beziehungsweise Linux befindet. Die Beschreibungen für Linux gelten im Großen und Ganzen auch für andere Unix-Varianten wie FreeBSD, Mac OS X oder Solaris; bei gravierenden Unterschieden finden Sie manchmal entsprechende Anmerkungen.

Für die beiden Systeme Linux und Windows wird die Installation und Grundkonfiguration folgender Komponenten beschrieben:

- Apache Webserver 2.2, Version 2.2.4
- MySQL-Server und -Tools; aktuelle Versionen 5.0.41 und 5.1.18 beta (die erste Alphaversion von 6.0 ist verfügbar, aber noch nicht einsatzreif)
- PHP 5.2 im Release 5.2.2
- phpMyAdmin, ein browserbasiertes MySQL-Administrationswerkzeug, in Version 2.10.1



Wenn Sie Ihre PHP/MySQL-Anwendungen in gemietetem Webspace bei einem Hoster betreiben möchten oder müssen, stehen Ihnen diese neuen Versionen in aller Regel nicht zur Verfügung. Üblich sind zurzeit oft noch Apache 1.3.x, PHP 4.4.x und MySQL 4.1.x oder gar nur 4.0.x. Deshalb wird in diesem Buch für die meisten

MySQL- und PHP-Aspekte erwähnt, ab welcher Version sie jeweils funktionieren. Wenn Sie auf Features neuerer Versionen angewiesen sind, bleibt Ihnen meist nichts anderes übrig, als bei Ihrem Hoster keinen einfachen Webspace, sondern einen virtuellen oder dedizierten Server zu mieten – in diesem Fall sind Sie aber auch selbst für dessen Sicherheit verantwortlich. Sie müssen ihn also sicher genug konfigurieren und regelmäßig Updates einspielen, weil Sie ansonsten unter Umständen zur Verantwortung gezogen werden, wenn Angreifer etwa Spam oder illegale Softwarekopien über Ihren Server verbreiten.

Sämtliche Software ist auf der beiliegenden CD-ROM enthalten. Vor der Installation sollten Sie aber jeweils überprüfen, ob eine neuere Version verfügbar ist, und diese gegebenenfalls herunterladen. Die Adressen der entsprechenden Websites finden Sie in Anhang D.

Auf Linux-Systemen gibt es drei grundsätzliche Formate für Software-Installationspakete:

- Quellcode-Archive zum Selbstkompilieren
- Archive mit Binärdateien und Installer-Skript
- spezielle Installationspakete in einem distributionsspezifischen Format

Jedes der Programme, deren Installation hier beschrieben wird, steht in einem oder mehreren dieser Formate zur Verfügung. In der Regel wird im Folgenden die Installation aus dem Quellcode beschrieben. Als Beispiel für eine distributionsbasierte Installation erfahren Sie weiter unten im Kasten »Distributionsware – LAMP unter openSUSE 10.2« auf Seite 55, welche Pakete Sie auf einem openSUSE 10.2-System für eine vollständige LAMP-Umgebung aktivieren müssen.

Wenn Sie auf einem Unix-System die Quellcode-Pakete der Hersteller installieren möchten, brauchen Sie einen ANSI-konformen C-Compiler. gcc eignet sich besonders gut, da er wahrscheinlich am häufigsten getestet wurde. Er ist in praktisch jeder Linux- oder Unix-Distribution enthalten, wird aber bei einer Standardinstallation oft nicht mitinstalliert. Bemühen Sie also gegebenenfalls den Paketmanager und die Hilfedateien Ihrer Distribution. Unter openSUSE 10.2 können Sie zum Beispiel YaST verwenden (siehe oben genannten Kasten), um mithilfe der Suchfunktion folgende Pakete zu finden und nachzuinstallieren:

- `gcc`
- `gcc-c++`
- `glibc-devel`
- `Flex`
- `libxml2-devel`

Für Windows-Rechner werden fast durchweg Installer-Pakete angeboten, die Ihnen per mehrseitigem Dialog den Weg durch die Installation weisen. Manche Software wird dagegen in einem ZIP-Archiv geliefert; in diesem Fall muss das Verzeichnis, in das Sie es entpacken, oft in eine Konfigurationsdatei eingetragen werden.



Den früheren Privatkunden-Windows-Versionen 95, 98, ME fehlt eine wichtige Eigenschaft für den Betrieb von Serversoftware, nämlich die Unterstützung sogenannter Dienste, die im Hintergrund ausgeführt werden. Wenngleich einige der hier besprochenen Serveranwendungen dies auf verschiedene Art kompensieren können, konzentrieren sich die Installationsanleitungen auf die Windows NT-Familie, das heißt Windows 2000, Windows XP, Windows Vista und Windows Server 2003. Für ein Produktivsystem sollten Sie ohnehin nur diese Systeme in Betracht ziehen; außer an Diensten mangelt es den alten Consumer-Systemen nämlich auch an Sicherheitsoptionen.

Eine interessante Alternative für Rechner, die noch keine der LAMP-/WAMP-Komponenten enthalten, ist das Komplettpaket XAMPP von *http://www.apachefriends.org*, das Sie ebenfalls auf der beiliegenden CD-ROM finden. Es enthält die für die Arbeit mit diesem Buch benötigten Komponenten Apache, MySQL, PHP und phpMyAdmin sowie weitere nützliche Serversoftware. Sie brauchen die Archivdatei für Ihr Betriebssystem (Windows[1], Linux oder Mac OS X) nur zu entpacken und können anschließend jeden Server manuell starten oder beenden. Beachten Sie aber, dass XAMPP ab Werk nicht sonderlich sicher konfiguriert ist und daher nur für Entwicklerrechner, aber nicht für Server im Interneteinsatz geeignet ist.

Zu Unrecht gefürchtet: das Arbeiten mit der Konsole

Zahlreiche Arbeitsschritte im Zusammenhang mit der Konfiguration Ihrer LAMP- oder WAMP-Umgebung, aber auch bei der späteren Administration von MySQL, benötigen eine Konsole, also ein Programm zur manuellen Befehlseingabe. Dies ist unter Windows die bereits erwähnte Eingabeaufforderung und auf Unix-Systemen ein beliebiges Terminalfenster. Zum Öffnen der Windows-Eingabeaufforderung gibt es zwei Möglichkeiten:

- Wählen Sie *Start* → *Alle Programme* → *Zubehör* → *Eingabeaufforderung*. Beachten Sie, dass es in älteren Windows-Versionen vor XP *Programme* statt *Alle Programme* heißt.
- Alternativ können Sie *Start* → *Ausführen* wählen, `cmd` eingeben und *OK* anklicken beziehungsweise Enter drücken.

1 Zurzeit läuft XAMPP noch nicht ohne Anpassungen unter Vista. Suchen Sie auf der Website des Anbieters nach entsprechenden Hinweisen.

Wie Sie in Ihrem Unix-artigen System ein Terminalfenster öffnen, ist je nach Distribution, Version und grafischer Oberfläche verschieden. Hier nur einige Beispiele:

- Der beliebte Desktop KDE für Linux und einige andere Unix-Varianten enthalten ein komfortables Terminalprogramm namens *Konsole*, das Sie in der Regel über ein Bildschirmsymbol im Panel (die Leiste am unteren Bildschirmrand) öffnen können.
- Der andere verbreitete Desktop, GNOME, besitzt ebenfalls eine eigene Terminalemulation, die einfach *GNOME Terminal* heißt. Sie öffnen es am einfachsten, indem Sie mit der rechten Maustaste eine leere Stelle auf dem Desktop anklicken und *Terminal öffnen* aus dem Kontextmenü wählen.
- Unter Mac OS X befindet sich das *Terminal* im Systemordner *Applications*. Wenn Sie ernsthaft mit dem Programmieren beginnen, werden Sie es öfter benötigen und sollten es daher ins Dock ziehen.



Beachten Sie, dass innerhalb von Unix-Terminalfenstern unterschiedliche *Shells* (Befehls-Interpreter) ausgeführt werden können, wodurch sich die Syntax mancher Eingaben etwas ändert. In den drei genannten Beispielfällen ist es so gut wie immer die `bash`. Soweit die Arbeit mit der Shell in diesem Abschnitt geschildert wird, macht dies aber keinen Unterschied.

Nachdem Sie die jeweilige Konsole geöffnet haben, wird ein *Prompt* (die eigentliche Eingabeaufforderung) angezeigt. Windows-Rechner verwenden standardmäßig die Schreibweise `Arbeitsverzeichnis\>`, zum Beispiel:

```
C:\Dokumente und Einstellungen\Sascha\Eigene Dateien>
```

Bei Unix-Systemen kann der Prompt sehr unterschiedlich aussehen. Recht häufig ist die Form `Username@Rechner:Arbeitsverzeichnis>`. Das eigene Home-Verzeichnis, meist `/home/Username`, wird dabei in der Regel durch ~ abgekürzt. Somit sieht der gesamte Prompt beispielsweise wie folgt aus:

```
sascha@linuxbox:~ >
```

Wenn Sie als `root` arbeiten, wird meistens kein Username angezeigt, und das Schlusszeichen wechselt von > oder `$` zu einer Raute (`#`), zum Beispiel:

```
linuxbox:/home/sascha #
```

In diesem Abschnitt und im Rest dieses Buchs werden normalerweise (solange der *konkrete* Prompt keine Rolle spielt) folgende Zeichen verwendet, um den Prompt zu kennzeichnen:

- >: Windows-Prompt sowie allgemeiner Prompt, wenn eine Eingabe für alle Betriebssysteme gilt.

- $: Unix-Prompt; beliebiger Benutzer einschließlich `root` (wobei Sie normale Aufgaben aus Sicherheitsgründen nicht als `root` erledigen sollten).
- #: Unix-Prompt für `root`.



Der weiter unten vorgestellte Kommandozeilenclient mysql zur manuellen Eingabe von SQL-Anweisungen verwendet den speziellen Prompt `mysql>`; dieser wird zur Unterscheidung stets mit abgedruckt.

Wenn Sie zum ersten Mal in einer Konsole arbeiten, werden Sie einige grundlegende Befehle benötigen. Diese betreffen vor allem den Umgang mit Verzeichnissen, wie etwa den Wechsel des Arbeitsverzeichnisses oder das Anlegen neuer Unterverzeichnisse. Hierbei spielt die unterschiedliche Organisation des Dateisystems, also die Verzeichnishierarchien, eine wichtige Rolle:

- Auf Windows-Rechnern beginnen Dateipfade mit einem Laufwerkbuchstaben, darauf folgen die ineinander verschachtelten Verzeichnisnamen und zum Schluss der Dateiname. Das Trennzeichen ist ein Backslash (\), der auf einer deutschen Windows-Tastatur mit Alt Gr + ß erzeugt wird. Das folgende Beispiel ist der Pfad der Textdatei *hello.txt* im Ordner *mydata* unter dem »Privatverzeichnis« des Users Sascha:

  ```
  C:\Dokumente und Einstellungen\Sascha\Eigene Dateien\mydata\hello.txt
  ```

- Unix-Systeme kennen keine Laufwerkbuchstaben. Das Dateisystem besitzt eine gemeinsame Wurzel namens `/`, wobei sich die diversen Standardverzeichnisse auf verschiedenen Laufwerken oder Partitionen befinden können – die genaue Anordnung wird durch Konfigurationsdateien geregelt. Als Trennzeichen zwischen Unterverzeichnissen sowie zwischen Verzeichnis und Datei dient dabei der Slash (`/`). Die Unix-Entsprechung des oben gezeigten Windows-Pfads wäre daher:

  ```
  /home/sascha/mydata/hello.txt
  ```



Beachten Sie, dass Unix bei Datei- und Verzeichnisnamen zwischen Groß- und Kleinschreibung unterscheidet, Windows aber nicht (daraus leiten sich, wie unten gezeigt, dieselben Regeln für MySQL-Datenbanken und -Tabellen ab). Tun Sie sich zur Sicherheit selbst einen Gefallen und schreiben Sie konsequent alles klein. Ebenso sollten Sie in allen selbst gewählten Namen Leerzeichen und Sonderzeichen (außer dem Unterstrich _) vermeiden. Spätestens wenn Sie Ihre Dateien im Web publizieren, kann es sonst zu Problemen kommen.

Manche Konsolenbefehle können mit Platzhaltern umgehen, die auf mehrere Dateien zutreffen. Dabei steht ein * für eine unbestimmte Anzahl beliebiger Zeichen und ein ? für genau ein beliebiges Zeichen. Auch hier gibt es einen kleinen Plattformunterschied: In Unix-Systemen steht * für alle Dateien in einem Verzeichnis, bei Windows dagegen `*.*`, weil die Dateiendung gesondert betrachtet wird.

Um das aktuelle Arbeitsverzeichnis zu wechseln, verwenden sowohl Unix als auch Windows das Kommando `cd` (kurz für »change directory«). Unter Windows ist dieser Befehl nicht dafür zuständig, das Laufwerk zu wechseln. Dies geschieht durch die einfache Eingabe des Laufwerkbuchstabens mit nachfolgendem Doppelpunkt. Das folgende Beispiel vollzieht einen Wechsel auf die Festplatte `C:`:

```
> C:
```

Mit `cd` können Sie unter Windows innerhalb eines Laufwerks einen absoluten, das heißt vollständigen Pfad angeben, um in das entsprechende Verzeichnis zu wechseln. Wenn Verzeichnis- oder Dateinamen Leerzeichen enthalten, müssen Sie diese (oder wahlweise den gesamten Pfad) in Anführungszeichen setzen. Das folgende Beispiel wechselt – aus einem beliebigen Verzeichnis auf der Festplatte `C:` – in das Verzeichnis *Eigene Dateien* des Benutzers Sascha:

```
> cd "\Dokumente und Einstellungen\Sascha\Eigene Dateien"
```

Bei Unix-Systemen funktioniert der Verzeichniswechsel per absolutem Pfad im Prinzip genauso, zum Beispiel:

```
$ cd /home/sascha
```

Wenn Sie vom aktuellen Verzeichnis aus in ein untergeordnetes Verzeichnis wechseln möchten, müssen Sie den Namen dieses Unterverzeichnisses ohne führenden Backslash beziehungsweise Slash angeben. Hier ein Windows-Beispiel:

```
C:\Dokumente und Einstellungen\Sascha\Eigene Dateien> cd mydata
C:\Dokumente und Einstellungen\Sascha\Eigene Dateien\mydata>
```

Auf diese Weise lassen sich auch mehrere Hierarchiestufen überwinden. Dazu sehen Sie hier ein Unix-Beispiel:

```
sascha@linuxbox:~ > cd mydata/briefe
sascha@linuxbox:~/mydata/briefe >
```

Um in das übergeordnete Verzeichnis zu wechseln, wird auf beiden Plattformen der spezielle Verzeichnisname `..` verwendet, zum Beispiel (unter Windows):

```
C:\Dokumente und Einstellungen\Sascha\Eigene Dateien> cd ..
C:\Dokumente und Einstellungen\Sascha>
```

Diese Techniken lassen sich kombinieren, um über sogenannte relative Pfade von jedem beliebigen Verzeichnis in jedes andere zu wechseln. Das folgende Beispiel vollzieht auf einem Unix-Rechner einen Wechsel aus dem oben gezeigten Verzeichnis *briefe* in das »Geschwister-Verzeichnis« *rechnungen*:

```
sascha@linuxbox:~/mydata/briefe > cd ../rechnungen
sascha@linuxbox:~/mydata/rechnungen >
```

Um unterhalb des aktuellen Arbeitsverzeichnisses ein neues Verzeichnis zu erstellen, wird das Kommando `mkdir` verwendet (Windows erlaubt auch die Kurzfassung `md`). Hier wird beispielsweise ein Verzeichnis für Termine angelegt:

```
sascha@linuxbox:~/mydata > mkdir termine
```

Beachten Sie, dass Sie auf einer Unix-Maschine lediglich innerhalb Ihres eigenen Home-Verzeichnisses neue Verzeichnisse erstellen dürfen. In anderen Bereichen des Dateisystems darf dies nur der Superuser `root`. Geben Sie `su` und das `root`-Passwort ein, wenn Sie vorübergehend als `root` arbeiten müssen, und `exit`, sobald Sie damit fertig sind. Sie können auch einen einzelnen Befehl als `root` ausführen, indem Sie

```
$ sudo Befehl Optionen ...
```

und anschließend das `root`-Passwort eingeben. Wenn Sie dies zum ersten Mal tun, wird der folgende Sicherheitshinweis angezeigt:

```
We trust you have received the usual lecture from the local System Administrator.
 It usually boils down to these three things:

#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.[2]
```

Als Nächstes sollten Sie noch das Kommando kennen, mit dem Sie sich den Inhalt des aktuellen Verzeichnisses ausgeben lassen können. Unter Windows heißt es `dir`:

```
> dir
```

Auf Unix-Rechnern lautet der Befehl dagegen `ls`. Wenn Sie die Option `-l` hinzufügen, erhalten Sie ausführliche Informationen über jede Datei – beispielsweise den Eigentümer, die Zugriffsrechte und die Größe:

```
$ ls -l
```

Um den Überblick zu behalten, ist es manchmal nützlich, den Fensterinhalt zu löschen und den Prompt wieder nach links oben zu setzen. Geben Sie dazu in der Windows-Eingabeaufforderung Folgendes ein:

```
> cls
```

In den meisten Unix-Terminals lautet der Befehl dagegen:

```
$ clear
```

Noch praktischer ist, dass Sie bei fast allen Unix-Varianten einfach Strg + L drücken können, um denselben Effekt zu erzielen.

Tabelle 2-2 stellt die wichtigsten Konsolenbefehle für beide Plattformen noch einmal gegenüber, wobei einige zusätzliche Anweisungen hinzukommen.

2 Übersetzung: »Wir vertrauen darauf, dass Sie die übliche Belehrung durch den lokalen Systemadministrator erhalten haben. Sie lässt sich für gewöhnlich durch folgende drei Punkte zusammenfassen: #1) Respektieren Sie die Privatsphäre anderer. #2) Denken Sie nach, bevor Sie tippen. #3) Große Macht bedingt große Verantwortung.«

Tabelle 2-1: Die wichtigsten Konsolenbefehle für Windows und Unix

Gewünschte Wirkung	Windows-Befehl	Unix-Befehl
Laufwerk wechseln	*Laufwerkbuchstabe*`:`, z.B. `C:` oder `F:`	–
Arbeitsverzeichnis wechseln – absoluter Pfad	`cd \Verz.[\Unterv.\...]`	`cd /Verz.[/Unterv./...]`
In Unterverzeichnis des aktuellen Arbeitsverzeichnisses wechseln	`cd Verz.[\Unterv.\...]`	`cd Verz.[/Unterv./...]`
In übergeordnetes Verzeichnis wechseln	`cd ..`	`cd ..`
In das eigene Home-Verzeichnis wechseln	–	`cd ~`
Neues Verzeichnis erstellen	`mkdir` *Name* `md` *Name*	`mkdir` *Name*
Inhalt des aktuellen Verzeichnisses anzeigen	`dir`	`ls` (ausführlich: `ls -l`)
Datei löschen	`del` *Name*	`rm` *Name*
Datei kopieren	`copy` *AltName NeuName*	`cp` *AltName NeuName*
Platzhalter: alle Dateien im aktuellen Verzeichnis	`*.*`	`*`
Bildschirm löschen	`cls`	`clear` (oft auch Strg + L)

Apache 2 installieren und konfigurieren

Der Apache-Webserver oder Apache HTTP Server ist das Kernstück Ihres LAMP-/WAMP-Systems. Er kommuniziert mit den Browsern der Benutzer und liefert ihnen unter anderem die Ausgabe von PHP-Skripten; umgekehrt kann er Benutzereingaben aus Webformularen wiederum an PHP-Skripten weiterleiten. Eine direkte Kommunikation mit der MySQL-Datenbank führt er dagegen in der Regel nicht durch.

Seit Apache 2.2 gibt es allerdings das optionale Modul `mod_dbd`, das über einen Treiber eine Verbindung zwischen dem Webserver und einer SQL-Datenbank herstellt; ein Treiber für MySQL ist separat zum Download verfügbar.[3] Die einzige mitgelieferte Nutzanwendung ist bisher das Modul `mod_authn_dbd`, das Vergleichspasswörter zur Authentifizierung per Datenbankabfrage einliest. Für die Zukunft sind aber durchaus weitere Module denkbar, die etwa Logdaten in eine Datenbanktabelle schreiben oder Webanwendungen einen abstrahierten Datenbankzugriff zur Verfügung stellen.

3 Unglücklicherweise erlaubt die Inkompatibilität zwischen Apache-Lizenz und GPL es nicht, den MySQL-Treiber mit der Apache-Distribution zu liefern.

Apache-Module

Eine der wichtigsten Eigenschaften von Apache ist sein modularer Aufbau. Fast jeder Aspekt seines Leistungsspektrums wurde als separates Modul realisiert, das Sie je nach Bedarf einkompilieren oder weglassen können. Mit der sogenannten DSO-Unterstützung (siehe Option `--enable-so` weiter unten) lassen sich die Module sogar bei jedem Start von Apache ein- oder ausschalten. Tabelle 2-2 zeigt eine Übersicht einiger wichtiger Module. Neben den Namen und Aufgaben der Module erfahren Sie auch, ob diese zum Lieferumfang von Apache 2 gehören und ob sie – in diesem Fall – bei einer Standardinstallation automatisch aktiviert werden. Unter *http://modules.apache.org* erhalten Sie zudem zahlreiche Drittanbietermodule. Mithilfe der Apache-API können einigermaßen erfahrene C-Programmierer sogar selbst Apache-Module schreiben.

Tabelle 2-2: Übersicht über einige wichtige Apache-Module

Modul	Bedeutung	mitgeliefert	aktiviert
`mod_authz_hostname`[a]	Host-basierte Zugriffskontrolle (Order/Allow/Deny, siehe unten)	ja	ja
`mod_alias`	Um- und Weiterleitung	ja	ja
`mod_auth_basic`	Klartextbasierte Benutzeranmeldung (Authentifizierung)	ja	ja
`mod_auth_digest`	Verschlüsselte Authentifizierung	ja	nein
`mod_authn_file`	Anmeldedaten aus Textdateien	ja	ja
`mod_authz_user`	Zugriffskontrolle für angemeldete Benutzer	ja	ja
`mod_authz_groupfile`	Zugriffskontrolle aufgrund von Gruppenzugehörigkeiten	ja	ja
`mod_autoindex`	Automatisch erstellter Verzeichnisindex	ja	ja
`mod_cgi`	CGI-Skripten (altmodische Webserver-Anwendungen)	ja	ja
`mod_dir`	Definition der Indexseite	ja	ja
`mod_log_config`	Protokollierung (Logdateien)	ja	ja
`mod_mime`	Zuweisungen von Dateityp, Zeichensatz und Sprache	ja	ja
`mod_negotiation`	Content-Negotiation (MIME-Varianten je nach Clientvorgabe)	ja	ja
`mod_perl`	In Apache integrierter Perl-Interpreter	nein	–
`mod_php`	In Apache integrierter PHP-Interpreter (siehe unten)	nein	–
`mod_ssl`	Verschlüsselte Verbindungen (https)	ja	nein

[a] Bis einschließlich Apache 2.0.x hieß dieses Modul `mod_access`.

Installation unter Linux

In vielen Linux-Distributionen ist der Apache-Webserver bereits ab Werk enthalten oder kann bei der Installation des Systems optional ausgewählt werden. Falls dies bei Ihnen der Fall ist, können Sie diesen Abschnitt überspringen – Sie sollten ledig-

lich sicherstellen, dass Apache 2.2 oder zumindest 2.0 installiert ist und keine 1.3-Version. Im Kasten »Distributionsware – LAMP unter openSUSE 10.2« auf Seite 55 wird beispielhaft die Aktivierung des mitgelieferten Apache-Webservers unter openSUSE 10.2 erläutert. Zu Ihrer eigenen Sicherheit sollten Sie außerdem überprüfen, ob die installierte Version aktuell ist – ältere Releases enthalten oft bekannte Sicherheitslücken, die sich Cracker zunutze machen.

Auf Unix-Systemen wird Apache fast immer aus dem Quellcode kompiliert; diese Installationsmethode wird von der Apache Software Foundation selbst empfohlen. Der erste Schritt besteht darin, dass Sie das Quellcode-Paket durch folgende Anweisung entpacken:

```
# tar -xzvf httpd-2.2.4.tar.gz
```

Wechseln Sie anschließend in das neu entstandene Verzeichnis, in das die Archivdateien entpackt wurden:

```
# cd httpd-2.2.4
```

Nun muss das Skript `configure` im aktuellen Verzeichnis aufgerufen werden, um die Quelldateien vor der eigentlichen Kompilierung an Ihre Bedürfnisse anzupassen. Das Skript kennt unzählige Optionen; geben Sie Folgendes ein, wenn Sie sie alle studieren möchten:

```
# ./configure --help |less
```

Hier nur die allerwichtigsten Optionen im Überblick:

- `--prefix=`*/Verzeichnispfad* gibt das übergeordnete Verzeichnis für die Apache-Installation an; Standard ist */usr/local/apache2*.
- `--enable-layout=`*Layoutname* ermöglicht alternativ (oder zusätzlich) zur Angabe eines Verzeichnisses die Auswahl eines Installationslayouts, das die Verzeichnisse für die verschiedenen Komponenten festlegt. Die Datei *config.layout* enthält die Definitionen der verschiedenen Layouts. Sie können sie mit einem Texteditor öffnen, um das für Sie passende Layout zu finden (oder nach dem Schema in der Datei selbst zu erstellen).
- `--with-mpm=`*MPM-Modul* wählt ein sogenanntes Multi-Processing-Modul (MPM) oder auch Laufzeitmodell aus: Eine der wichtigsten Neuerungen von Apache 2 besteht darin, dass er verschiedene, für unterschiedliche Plattformen optimierte Methoden zur Verarbeitung mehrerer gleichzeitiger Clientverbindungen kennt. Unter Unix wird standardmäßig das MPM `prefork` installiert, das für jede Verbindung einen eigenen Prozess einsetzt. Moderne Linux-Systeme unterstützen neben Prozessen auch die speicherschonenden Threads. Wenn Sie keine exotischen Module verwenden, kann sich deshalb die Einstellung `--with-mpm=worker` lohnen – sie installiert das Thread-basierte Worker-MPM. Für Windows und andere Nicht-Unix-Systeme gibt es übrigens spezielle MPMs.



Die Entwickler von PHP empfehlen, PHP in einer Produktivumgebung nicht mit Thread-MPMs einzusetzen. Das Problem dabei ist, dass einige PHP-Erweiterungen nicht Thread-sicher sind. Um Schwierigkeiten zu vermeiden, sollten Sie Apache 2 auf Unix-Systemen mit dem rein prozessbasierten MPM `prefork` betreiben.

- `--enable-so` ist eine der wichtigsten Optionen: Sie schaltet die Unterstützung für Dynamic Shared Objects (DSOs) ein, so dass Module dynamisch hinzugefügt werden können. Da Apache fast alle seine Aufgaben durch einzelne Module löst, ist dies praktisch unabdingbar; glücklicherweise ist diese Option inzwischen Standard.
- `--enable-modules="modul1 modul2 ..."` kompiliert die angegebenen Module fest ein. In der Liste müssen Sie das Kürzel `mod_` vor dem Modulnamen weglassen – also zum Beispiel `"autoindex log_config"` statt `"mod_autoindex mod_log_config"`. Für manche häufig genutzten Module steht übrigens auch die Kurzfassung `enable-modulname` zur Verfügung, zum Beispiel `enable-rewrite` für `mod_rewrite`.
- `--enable-mods-shared="modul1 modul2 ..."` kompiliert die angegebenen Module als DSOs, was den praktischen Vorteil hat, dass sie jederzeit (auch vorübergehend) wieder entfernt werden können. Auch hier gibt es für prominente Module die Schreibweise `enable-modulname=shared`.
- `--enable-modules=most` beziehungsweise `enable-mods-shared=most` kompiliert fast alle Module. Je nach Auswahl werden sie statisch einkompiliert oder als DSOs erstellt. `most` ist gegenüber `all` vorzuziehen, weil unter anderem alle Module weggelassen werden, die beim Kompilieren Probleme bereiten würden.
- `--disable-modules="modul1 modul2 ..."` lässt die angegebenen Module ausdrücklich weg. Das ist besonders beim Einsatz von `enable-modules=most` sehr nützlich.

Das folgende Beispiel bereitet Apache für die Kompilierung und Installation im Verzeichnis */usr/local/apache2* und zur Installation der meisten Module als DSOs vor:

```
# ./configure --enable-layout=Apache --enable-so \
--enable-mods-shared=most
```

Nachdem `configure` seine Arbeit beendet hat, die viele Minuten dauern kann, werden die beiden Befehle zur Kompilierung und Installation aufgerufen; zumindest für Letzteren benötigen Sie `root`-Rechte:

```
# make
# make install
```

Nach der Installation können Sie Apache starten. Dafür ist das Skript *apachectl* im *bin*-Verzeichnis Ihrer Installation (Standard: */usr/local/apache2/bin*) zuständig. Zunächst sollten Sie einen Symlink erzeugen, der dieses Skript in einem Verzeichnis

innerhalb Ihres PATH verfügbar macht, damit Sie sich die Pfadangabe in Zukunft sparen können. Eine gute Wahl ist beispielsweise:

```
# ln -s /usr/local/apache2/bin/apachectl /usr/bin/apachectl
```

Danach können Sie dieses Skript aus jedem beliebigen Verzeichnis heraus wie folgt benutzen, um Apache zu starten:

```
# apachectl start
```

Dieses Skript dient auch dazu, Apache nach einer Konfigurationsänderung neu zu starten. Geben Sie dazu Folgendes ein:

```
# apachectl restart
```

Bei Servern im Praxiseinsatz sollten Sie laufende Clientverbindungen nicht durch einen Neustart abbrechen, sondern zunächst abwarten, bis sie abgeschlossen sind. Das erledigt die Option `graceful` (»elegant«):

```
# apachectl graceful
```



Für manche Konfigurationsänderungen genügt ein `graceful`-Neustart nicht, und für einige wenige müssen Sie Apache sogar ganz beenden (`apachectl stop`) und wieder starten – beispielsweise nach der weiter unten beschriebenen Installation von PHP.

Zu guter Letzt kann `apachectl` auch dazu genutzt werden, Apache 2 beim Booten automatisch zu starten. In den meisten Linux-Distributionen können Sie dazu einen Symlink auf dieses Skript in */etc/init.d* ablegen und es anschließend mittels `chkconfig -a` aktivieren. Beispiel:

```
# ln -s /usr/local/apache2/bin/apachectl /etc/init.d/apache22
# chkconfig -a apache22
```



Der Dateiname *apache22* wurde hier absichtlich gewählt, um der eventuell installierten Apache-Version aus der Distribution, deren Startskript meist *apache2* heißt, nicht in die Quere zu kommen.

Installation unter Windows

Für Windows bietet die Apache Software Foundation einen bequemen Binär-Installer im MSI-Format (Windows Installer) an. Starten Sie das Paket *apache_2.2.4-win32-x86-no_ssl.msi* unter Windows 2000, XP oder Server 2003 per Doppelklick.[4]

Unter Windows Vista funktioniert diese Startvariante nicht, da Sie so keine Administratorrechte für den Installer erhalten. Hier müssen Sie zunächst die Eingabeauf-

4 Bei älteren Windows-Versionen als den angegebenen müssen Sie zuerst den Windows Installer selbst installieren. Suchen Sie einfach auf der Website von Microsoft (*http://www.microsoft.com*) nach diesem Begriff.

forderung als Administrator öffnen, indem Sie *Start* → *Alle Programme* → *Zubehör* wählen, den Eintrag *Eingabeaufforderung* mit der rechten Maustaste anklicken und *Als Administrator ausführen* wählen. Bewegen Sie sich danach mithilfe von `cd`-Befehlen (siehe den ersten Abschnitt dieses Kapitels) in das Verzeichnis mit der MSI-Datei und geben Sie Folgendes ein, um sie zu starten:

```
> msiexec /i apache_2.2.4-win32-x86-no_ssl.msi
```

Folgen Sie anschließend in jedem Fall Schritt für Schritt den Installationsanweisungen auf den folgenden Dialogseiten:

1. Information, dass Apache 2 installiert wird. Klicken Sie hier – wie auch auf allen folgenden Bildschirmen – *Next* an, um fortzufahren.
2. Anzeige der Apache-Lizenz. Es handelt sich um eine GPL-ähnliche Open Source-Lizenz; im Wesentlichen erlaubt sie den freien Einsatz und die Veränderung von Apache und schützt die Entwickler vor der Vereinnahmung durch kommerzielle Unternehmen oder gar Softwarepatente. Wählen Sie *I accept the terms in the license agreement*.
3. Kurze Übersicht über die Features von Apache 2 und weitere Informationsquellen.
4. Eingabe einiger Grundeinstellungen: *Network Domain* ist Ihr Domainname; für einen lokalen Testserver können Sie zum Beispiel *test.local* eingeben. *Server Name* ist der Name des Webservers, zum Beispiel *www.test.local*. Die *Administrator's E-Mail Address* wird angegeben, damit Benutzer Ihnen bei Fehlern eine entsprechende Mitteilung senden können; normalerweise wird dafür die Adresse *webmaster@<Ihre-Domain>* verwendet. Als Letztes müssen Sie auswählen, ob Apache als automatisch startender Produktionsserver (*For All Users, on Port 80, as a Service*) oder als manuell zu startendes Testprogramm (*only for the Current User, on Port 8080, when started Manually*) installiert werden soll. Selbst für lokale Testzwecke lohnt sich Ersteres; der Dienst verbraucht kaum Ressourcen und kann leicht gegen Zugriffe von außen geschützt werden (siehe unten).
5. Nun müssen Sie die Vollständigkeit der Installation auswählen: *Typical* installiert die wichtigsten Komponenten; mit *Custom* können Sie weitere Elemente auswählen. Sie sollten auf jeden Fall *Custom* einstellen, da Sie nur so den Installationsort bestimmen können.
6. Diese Dialogseite wird nur angezeigt, wenn Sie im vorigen Schritt *Custom* gewählt haben. Oben sehen Sie in einer Baumansicht die Bestandteile der Installation. Wenn genügend Festplattenspeicher zur Verfügung steht (etwa 100 MByte), sollten Sie einfach den obersten Eintrag anklicken und *This Feature, and all subfeatures, will be installed on local hard drive* wählen; andernfalls können Sie beispielsweise die Dokumentation weglassen. Unten können Sie das Installationsverzeichnis wählen – voreingestellt ist *Apache Software Foundation* im *Programme*-Verzeichnis.

7. Klicken Sie im letzten Bildschirm auf *Install*, um die Installation mit den gewählten Einstellungen zu starten, oder auf *Back*, falls Sie doch noch etwas ändern möchten.

Nach der Installation als Dienst wird Apache automatisch gestartet. Im Systray (rechter Rand der Taskleiste) erscheint das kleine Feder-Icon des *Apache Service Monitor*. Klicken Sie es mit der rechten Maustaste an, um den Monitor zu öffnen. Hier können Sie Apache steuern – zum Beispiel nach einer Konfigurationsänderung neu starten.

In Windows Vista funktioniert der Apache Monitor nicht. Entfernen Sie ihn daher zunächst aus dem Autostart, indem Sie *Start* → *Alle Programme* → *Autostart* wählen, ihn mit der rechten Maustaste anklicken und *Entfernen* wählen. Zur Steuerung von Apache müssen Sie unter Vista stattdessen die Diensteverwaltung verwenden, die Sie unter *Start* → *Systemsteuerung* → *Verwaltung* → *Dienste* finden.

Basiskonfiguration

Nun ist Apache installiert, und Sie können sich um die Grundkonfiguration kümmern, bevor Sie ihn einsetzen. Bei einer Apache-Standardinstallation befinden sich sämtliche Einstellungen in einer einzigen Konfigurationsdatei namens *httpd.conf*, die sich im *conf*-Verzeichnis des Servers befindet. Welches Verzeichnis dies unter Linux bei Ihrem Installationslayout ist, können Sie in der Datei *config.layout* im Eintrag `sysconfdir` ermitteln. Bei der Installation in */usr/local/apache2* mit dem Standardlayout `Apache` handelt es sich um das Verzeichnis */usr/local/apache2/conf*. Unter Windows befindet sich die Datei im Verzeichnis *<Apache-Basisverzeichnis>\conf*, bei einer Standardinstallation also unter *C:\Programme\Apache Software Foundation\Apache2.2\conf*.

Die Konfigurationsdatei besteht aus zahlreichen Konfigurationseinstellungen, den *Direktiven*. Die meisten von ihnen werden in der Datei selbst durch Kommentare beschrieben – ein Kommentar ist jede Zeile, die mit # anfängt. Eine deutschsprachige Referenz aller Direktiven finden Sie im offiziellen Apache Manual (*http://httpd.apache.org/docs-2.2/de/*), das normalerweise mit Apache auf Ihrem Rechner installiert wird.

Im Folgenden werden nur einige der wichtigsten Direktiven im Überblick vorgestellt, und zwar in der Reihenfolge, in der sie in der Standardkonfigurationsdatei einer Apache-Neuinstallation auftreten. Beachten Sie, dass es noch weitere wichtige Einstellungen gibt; diese betreffen allerdings die Sicherheit und Stabilität des Servers, so dass Sie die Voreinstellungen beibehalten sollten, falls Sie nicht ganz genau wissen, was Sie tun.



Die Werte vieler Direktiven sind Pfade. Beachten Sie, dass Sie auch unter Windows den Unix-Slash (`/`) als Trennzeichen benutzen müssen. Zudem müssen Pfade und andere Werte in Anführungszeichen stehen, sofern sie Leerzeichen enthalten.

`ServerRoot`

Diese Direktive gibt das Apache-Installationsverzeichnis an; in der Regel wurde es beim Installationsprozess bereits richtig eingestellt. Beispiel:

```
ServerRoot /usr/local/apache2
```

`Listen`

Diese Direktive stellt den *TCP-Port* ein, an dem Apache lauscht. Ports werden verwendet, um zu bestimmen, welche Anwendungen auf zwei Rechnern in einer Netzwerkverbindung miteinander kommunizieren. Serverdienste verwenden dabei in der Regel feste Portnummern zwischen 0 und 1023, während Clients beliebige höhere Nummern einsetzen. Das HTTP-Protokoll hat den Standardport 80, in der Regel steht hier also Folgendes:

```
Listen 80
```

Optional kann eine Netzwerkadresse angegeben werden, wenn Apache nicht an allen Netzwerkschnittstellen lauschen soll, sondern zum Beispiel nur im Intranet. Beispiel:

```
Listen 192.168.0.2:80
```

Falls Apache noch an anderen Ports lauschen soll (etwa 443 für gesicherte SSL-Verbindungen), werden weitere `Listen`-Direktiven benötigt.

`LoadModule`

Zum dynamischen Laden gewünschter DSO-Module müssen Sie `LoadModule` verwenden. Die Syntax lautet:

```
LoadModule xxx_module modules/mod_xxx.so
```

Das erste Argument ist die allgemeine Modulbezeichnung, das zweite der Pfad der Moduldatei. Letzterer kann – wie hier – relativ zu `ServerRoot` oder auch absolut sein. Das folgende Beispiel lädt das Modul `mod_autoindex` zur automatischen Erzeugung von Verzeichnisindizes:

```
LoadModule autoindex_module modules/mod_autoindex.so
```

`ServerAdmin`

Hier sollte bei öffentlichen Webservern die E-Mail-Adresse des Administrators (meist *webmaster@<Ihre-Domain>*) angegeben werden. Wenn der Apache automatisch Seiten erzeugt – vor allem für Fehlermeldungen wie »Seite nicht gefunden« –, kann er einen Link auf diese Adresse anzeigen, damit Benutzer entsprechende Fehlermitteilungen versenden können. Beispiel:

```
ServerAdmin webmaster@test.local
```

`ServerName`

Mithilfe von `ServerName` wird der Netzwerkname des Webservers angegeben. Beispiel:

```
ServerName www.test.local
```

Beachten Sie, dass diese Einstellung nur der Selbstidentifikation des Servers dient. Damit er tatsächlich unter diesem Namen im Netzwerk erreichbar ist, müssen Sie einen entsprechenden Eintrag auf einem Nameserver vornehmen. Für den Hausgebrauch genügt es auch, eine Zeile wie die folgende in die Datei */etc/hosts* (Windows: *<Windows-Verzeichnis>\System32\drivers\etc\hosts*) aller beteiligten Rechner zu schreiben:

```
192.168.0.2 www.test.local
```

Statt 192.168.0.2 müssen Sie natürlich die richtige IP-Adresse Ihres Webserver-Rechners angeben.



Wenn Sie Webserver und Browser auf demselben Rechner ausführen, ist der Server stets auch unter der speziellen IP-Adresse 127.0.0.1 (die *Loopback*-Adresse für »Selbstgespräche«) erreichbar; meist ist ihr auch der besondere Hostname *localhost* zugeordnet. Geben Sie also im Browser *http://127.0.0.1* oder *http://localhost* ein.

`DocumentRoot`

Hier wird das Stammverzeichnis der Website angegeben. Bei einer Unix-Standardinstallation sieht der Eintrag beispielsweise so aus:

```
DocumentRoot /usr/local/apache2/htdocs
```

Eine Clientanfrage mit der URL *http://www.test.local/products/info.html* würde in diesem Fall auf die Datei */usr/local/apache2/htdocs/products/info.html* verweisen. Hier noch ein Windows-Beispiel:

```
DocumentRoot "C:/Programme/Apache Software Foundation/Apache2.2/htdocs"
```

Beachten Sie in diesem Zusammenhang die Anführungszeichen – sie sind bei Apache-Direktiven zwingend erforderlich, sobald ein Parameter Leerzeichen enthält.

`<Directory> ... </Directory>`

Dies ist eine von mehreren sogenannten Container-Direktiven: Die darin enthaltenen Einstellungen beziehen sich nur auf das angegebene Verzeichnis und seine Unterverzeichnisse. Das wichtigste Beispiel sind die weiter unten gezeigten Voreinstellungen für alle Verzeichnisse und für die `DocumentRoot`; Letzteres sieht bei einer Unix-Standardinstallation schematisch so aus:

```
<Directory /usr/local/apache2/htdocs>
   # Voreinstellungen für die DocumentRoot ...
</Directory>
```

`<Location> ... </Location>`

Ähnelt `<Directory>` sehr, mit dem Unterschied, dass sich die Angabe auf einen URL-Pfad bezieht. Betrachten Sie dazu das folgende Beispiel:

```
<Location /info>
  # Einstellungen für URL-Pfad /info
</Location>
```

Die Direktiven in diesem Container betreffen also (auf das obige Beispiel bezogen) alle URLs, die mit *http://www.test.local/info/...* beginnen.

`Options`

`Options` stellt verschiedene Einstellungen für das Verhalten eines Verzeichnisses zur Verfügung. Beispiele: `Indexes` generiert automatisch eine Liste mit Hyperlinks auf die Dateien im Verzeichnis, wenn keine Indexseite (`DirectoryIndex`, siehe unten) gefunden wird; `Includes` aktiviert die Ausführung von SSI (Server Side Includes); `FollowSymLinks` löst symbolische Links (Verweise auf Dateien in anderen Verzeichnissen) auf. Beispiel:

```
Options Indexes FollowSymLinks
```

`AllowOverride`

Innerhalb der Website selbst können zusätzliche Dateien mit Konfigurationsdirektiven stehen, die nur das jeweilige Verzeichnis betreffen. Der Standardname einer solchen Datei lautet *.htaccess*. (Dieser Name kann auf Wunsch mithilfe der hier nicht näher beschriebenen Direktive `AccessFileName` geändert werden.) Die Direktive `AllowOverride` legt fest, ob – und welche – Direktiven in *.htaccess*-Dateien innerhalb der Verzeichnisse der Website selbst überschrieben werden dürfen. Neben `All` (jede grundsätzlich verzeichnisfähige Direktive) und `None` (keine) können einige Gruppen angegeben werden, zum Beispiel `FileInfo` (Dateieinstellungen) oder `AuthConfig` (Authentifizierung und Zugriffskontrolle). Beispiel:

```
AllowOverride FileInfo
```

Wenn Sie eine einzelne Website betreiben, die nur von Ihnen administriert wird, sollten Sie *.htaccess*-Dateien mittels `AllowOverride None` abschalten und sämtliche Konfigurationseinstellungen in die *httpd.conf* schreiben. Diese Dateien sind nur nützlich, wenn Sie die Verantwortung für einzelne Verzeichnisse an Dritte delegieren möchten.

`Order`, `Allow`, `Deny`

Diese drei Direktiven bestimmen, welche Hosts oder Netzwerke Zugriff auf ein bestimmtes Verzeichnis erhalten sollen. Bei `Allow` können Sie Hosts auflisten, denen der Zugriff explizit gewährt werden soll; `Deny` verbietet ihn grundsätzlich. `Order` kann die Werte `Allow,Deny` oder `Deny,Allow` (jeweils ohne Leerzeichen!) annehmen und bestimmt so, in welcher Reihenfolge `Allow`- und `Deny`-Angaben ausgewertet werden. In aller Regel wird eine der beiden Direktiven für

alle Hosts gesetzt (`Deny from all` beziehungsweise `Allow from all`); die jeweils andere definiert dann Ausnahmen. Hier ein Beispiel:

```
# URL-Pfad /lokal nur für das lokale Netz
<Location /lokal>
   Order Deny,Allow
   Deny from all
   Allow from 192.168.0
</Location>
```

Natürlich müssen Sie `192.168.0` durch den Netzwerkteil der IP-Adressen Ihres eigenen Netzes ersetzen. Beachten Sie im Übrigen, dass diese Direktiven normalerweise nichts bringen, wenn Sie bestimmte Benutzer ausschließen wollen, da die meisten IP-Adressen dynamisch von den Internetprovidern zugewiesen werden.

In den Verzeichniseinstellungen der `DocumentRoot` befinden sich bei öffentlichen Webservern folgende Einträge, die allen Hosts den Zugriff gewähren:

```
Order Allow,Deny
Allow from all
```

In einer reinen Testumgebung sollten Sie die Zugriffe allerdings auf den Rechner selbst (`127.0.0.1`) beziehungsweise auf das lokale Netzwerk beschränken.

`DirectoryIndex`

Diese Direktive bestimmt, welche Dateien als *Indexdokumente* (oder Startseiten) gelten sollen. Wenn ein Benutzer ein Verzeichnis, aber keine bestimmte Datei anfordert, sucht Apache automatisch nach Dateien mit den hier angegebenen Namen; die erste gefundene Datei wird ausgeliefert. Die Voreinstellung ist *index.html*; auf einem PHP-fähigen Webserver empfiehlt sich dagegen:

```
DirectoryIndex index.html index.php
```

Falls Sie die Dateiendung *.htm* statt *.html* bevorzugen, können Sie auch *index.htm* hinzufügen. Gegebenenfalls sind andere Namen als *index* ebenfalls möglich, etwa *start*, *home* oder *default*.

Was passiert, wenn im angesprochenen Verzeichnis gar keine Indexseite gefunden wird, hängt von weiteren Einstellungen ab: Sofern die Option `Indexes` (siehe oben) gesetzt ist, generiert Apache selbst einen Verzeichnisindex. Andernfalls wird die Statusmeldung »404 (Seite nicht gefunden)« an den Client gesendet.

`Alias`

Die Direktive `Alias` ordnet einem Verzeichnis außerhalb der `DocumentRoot` einen URL-Pfad zu. Auf diese Weise können Sie über den Webserver auch Dateien veröffentlichen, die aus organisatorischen Gründen oder aus Sicherheitserwägungen nicht im eigentlichen Website-Verzeichnis liegen. Betrachten Sie dazu folgendes Beispiel:

```
Alias /phpmyadmin /usr/local/phpMyAdmin-2.10.1
```

Das weiter unten behandelte Datenbank-Administrationstool phpMyAdmin, das in diesem Beispiel in Wirklichkeit im Verzeichnis */usr/local/phpMyAdmin-2.10.1* liegt, wird unter dem URL-Pfad */phpmyadmin* eingebunden. Es kann also unter der URL *http://www.test.local/phpmyadmin* aufgerufen werden.



Die Beispiele im weiteren Verlauf dieses Buchs gehen davon aus, dass folgende besonderen Apache-Konfigurationseinstellungen vorgenommen werden:

1. `DirectoryIndex` enthält die Dateinamen *index.html* und *index.php*.
2. Der Server ist unter dem Domainnamen *www.test.local* erreichbar (Sie müssen die Direktive `ServerName` entsprechend setzen und Ihre */etc/hosts*-Datei anpassen; siehe oben).
3. In allen Verzeichnissen, die Apache veröffentlicht, kann PHP ausgeführt werden. Die entsprechende Konfiguration wird weiter unten im Rahmen der PHP-Installation erläutert.

MySQL installieren

Der zweite Schritt ist die Installation des MySQL-Datenbankservers selbst. Dazu sind unter beiden Systemfamilien einige Schritte erforderlich. Wenn Sie der nachfolgenden ausführlichen Anleitung folgen, werden Sie aber keine Schwierigkeiten haben. Als konkrete Version wird hier 5.0.x beschrieben; die Installation der bisherigen Beta-Versionen von MySQL 5.1 funktioniert aber genauso.



Von Zeit zu Zeit ändern sich einige Details der MySQL-Installation, vor allem unter Windows. Falls es gravierende Änderungen gibt, werden diese auf der Website zum Buch (*http://buecher.lingoworld.de/mysql*) erläutert.

Installation unter Linux

Beachten Sie, dass Sie den Installationsprozess als `root` durchführen müssen, weil Sie Schreibrechte in systemrelevanten Verzeichnissen benötigen und mit Benutzer- und Gruppen-IDs umgehen müssen.

Für Linux (und einige andere Unix-Systeme) werden fertig kompilierte Binärpakete von MySQL angeboten. Die zum Zeitpunkt der Drucklegung neuesten Versionen sind auf der CD zum Buch enthalten; sehen Sie aber besser auf der MySQL-Website nach, ob nicht eine neuere Version verfügbar ist. Die entsprechende Archivdatei müssen Sie zunächst entpacken (hier als Beispiel die `max`-Variante):

```
# tar -xzvf mysql-max-5.0.41-linux-i686-glibc23.tar.gz
```



Hier und in den nachfolgenden Schritten brauchen Sie den elend langen Dateinamen wahrscheinlich nicht auszuschreiben, sondern können zum Beispiel `mysql-` eintippen und dann Tab drücken, um von der Eingabevervollständigung Ihres Terminalfensters Gebrauch zu machen.

Das ausgepackte Verzeichnis sollten Sie nach */usr/local* verschieben, falls Sie es nicht bereits dort entpackt haben:

```
# mv mysql-max-5.0.41-linux-i686-glibc23 /usr/local
```

Wechseln Sie nun nach */usr/local*:

```
# cd /usr/local
```

Als Nächstes sollten Sie einen symbolischen Link erstellen, der den umständlichen Verzeichnisnamen mit der vollständigen Versionsbezeichnung unter dem praktischeren Namen *mysql* bereitstellt:

```
# ln -s mysql-max-5.0.41-linux-i686-glibc23 mysql
```

Alternativ zu den bisherigen Schritten können Sie auch das Quellcode-Paket kompilieren, das manchmal in einem aktuelleren Release vorliegt als die Binärvariante. Entpacken Sie es zunächst:

```
# tar -xzvf mysql-5.0.41.tar.gz
```

Mittels `cd` können Sie in das neu erstellte Verzeichnis wechseln:

```
# cd mysql-5.0.41
```

Geben Sie Folgendes ein, um Auskunft über alle verfügbaren Konfigurationsoptionen zu erhalten:

```
# ./configure --help |less
```

Die Standardauswahl der Optionen ist in vielen Fällen in Ordnung; es genügt also meistens, dieses einzugeben, um den Quellcode zur Installation unter */usr/local/mysql* vorzubereiten:

```
# ./configure --prefix=/usr/local/mysql
```

Nachdem `configure` seine Arbeit beendet hat, werden die üblichen Befehle zur Kompilierung und Installation aufgerufen:

```
# make
# make install
```

Wenn Sie den bisherigen Anweisungen gefolgt sind und entweder die Binärdistribution entpackt oder das Quellcode-Paket kompiliert haben, befindet sich MySQL unter */usr/local/mysql*. Wechseln Sie als Nächstes in dieses Verzeichnis:

```
# cd /usr/local/mysql
```

Erstellen Sie eine Gruppe und einen zu dieser Gruppe gehörigen Benutzer, unter deren ID MySQL ausgeführt werden soll:

```
# groupadd mysql
# useradd -g mysql mysql
```

Der nächste Schritt besteht darin, das Installer-Skript ausführen. Es erstellt vor allem die Grunddatenbanken und -tabellen, beispielsweise die Grant-Tabellen mit den MySQL-Benutzern und -Berechtigungen. Geben Sie Folgendes ein:

```
# scripts/mysql_install_db --user=mysql
```

Damit der Datenbankserver funktioniert, müssen Sie nun einige Besitzrechte setzen. Grundsätzlich müssen alle Dateien im *mysql*-Verzeichnis und in allen Unterverzeichnissen `root` gehören (die Option `-R` steht für rekursiv, das heißt, sie durchläuft den gesamten Verzeichnisbaum):

```
# chown -R root .
```

Das Verzeichnis *data* muss dagegen dem *mysql*-Benutzerkonto gehören, weil es hier Schreibrechte benötigt:

```
# chown -R mysql data
```

Für allgemeine Leserechte wird der gesamte Inhalt des Verzeichnisses schließlich der Gruppe `mysql` überschrieben:

```
# chgrp -R mysql .
```

Nachdem nun alle Installations- und Grundkonfigurationsarbeiten abgeschlossen sind, können Sie den Datenbankserver starten:

```
# bin/mysqld_safe --user=mysql &
```

Das angehängte `&` sorgt dafür, dass die Anweisung im Hintergrund ausgeführt wird. Auf diese Weise können Sie das entsprechende Terminalfenster weiterverwenden, während der MySQL-Server ausgeführt wird.

Als Nächstes sollten Sie zur künftigen Arbeitserleichterung wieder einige symbolische Links erstellen, um wichtige MySQL-Dienstprogramme von jedem Verzeichnis aus zur Hand zu haben. Insbesondere der Kommandozeilenclient *mysql*, das Verwaltungswerkzeug *mysqladmin* und das Backup-Tool *mysqldump* sollten stets verfügbar sein:

```
# ln -s /usr/local/mysql/bin/mysql /usr/bin/mysql
# ln -s /usr/local/mysql/bin/mysqladmin /usr/bin/mysqladmin
# ln -s /usr/local/mysql/bin/mysqldump /usr/bin/mysqldump
```

Zu guter Letzt können Sie auch den MySQL-Server so einrichten, dass er automatisch gestartet wird. Als Startskript dient dabei *mysql.server* im Unterverzeichnis *support-files* Ihrer MySQL-Installation. Wohin Sie dieses Skript verknüpfen müssen und wie Sie es aktivieren, kann sich je nach Linux-Distribution unterscheiden;

schlagen Sie gegebenenfalls im Handbuch Ihres Systems nach. Unter openSUSE und vielen anderen Distributionen funktioniert folgende Befehlsfolge:

```
# ln -s /usr/local/mysql/support-files/mysql.server /etc/init.d/mysql5
# chkconfig -a mysql5
```

Installation unter Windows

Der Standard-Installer für Windows ist eine ZIP-Datei namens *mysql-5.0.41-win32.zip*. Auf der MySQL-Website wird auch ein kleineres Paket namens *Windows Essentials* (Dateiname zurzeit *mysql-essential-5.0.41-win32.msi*) angeboten, das allerdings nur die unbedingt für den Betrieb eines MySQL-Servers erforderlichen Komponenten enthält. Wenn genügend Festplattenplatz vorhanden ist (was auf einem Rechner, auf dem Serverdienste ausgeführt werden sollen, selbstverständlich sein sollte), ist es besser, das hier beschriebene *Complete*-Paket zu installieren, damit Sie für alle erdenklichen Aufgaben gerüstet sind.

Nachdem Sie die Datei von der CD kopiert oder aus dem Web heruntergeladen haben, können Sie sie – beispielsweise mit WinZip – entpacken. Sie enthält ausschließlich die Datei *Setup.exe*. Diese wird unter Windows 2000, XP, Server 2003 und so weiter per Doppelklick ausgeführt. Auf Vista-Systemen müssen Sie sie dagegen mit der rechten Maustaste anklicken und *Als Administrator ausführen* wählen. Danach müssen Sie gegebenenfalls noch eine Sicherheitsabfrage bestätigen.

Der Installer stellt Ihnen auf mehreren Registerkarten einige Fragen; mithilfe der Schaltfläche *Next* kommen Sie jeweils weiter:

1. *Welcome* – kurze Information darüber, dass MySQL installiert wird.
2. *Setup Type* – wählen Sie am besten *Custom*, um selbst die Komponenten auszuwählen, die Sie installieren möchten. *Typical* installiert eine Standardauswahl; mit *Complete* wird dagegen ohne weitere Nachfrage alles installiert.
3. Falls Sie im vorigen Schritt *Custom* gewählt haben, erscheint nun die Registerkarte *Custom Setup*. In einer Baumstruktur können Sie die Komponenten oder Unterkomponenten auswählen, die Sie installieren beziehungsweise weglassen möchten. Den *MySQL Server* benötigen Sie dabei auf jeden Fall – es sei denn, Sie möchten nur die Client-Tools installieren, um über das Netzwerk den MySQL-Server auf einem anderen Server zu administrieren. Die *Client Programs* besitzen drei Unterpunkte: Die *MySQL Command Line Shell* ist der Kommandozeilenclient (siehe Kapitel 4), die *MySQL Command Line Utilities* sind zusätzliche Administrationstools, die vor allem in Kapitel 9 beschrieben werden, und die auf den nächsten Seiten behandelte *MySQL Server Instance Config* dient der Einrichtung des Servers nach der Installation. Alle drei Komponenten sind für den erfolgreichen Betrieb eines MySQL-Servers (und zum Durcharbeiten dieses Buchs) unerlässlich. Der *MySQL Instance Manager* ist ein

neueres Kommandozeilentool, das ebenfalls in Kapitel 9 angesprochen wird. Die *Documentation* (die vollständige MySQL-Dokumentation in diversen Dateiformaten) können Sie weglassen, wenn Sie Platz sparen möchten – sie ist jederzeit online unter *http://dev.mysql.com/doc/mysql/* verfügbar. Die *Developer Components* schließlich werden seltener benötigt – sie dienen speziellen Entwicklungsaufgaben: Die *C Include Files / Lib Files* ermöglichen den Zugriff auf MySQL-Datenbanken aus eigenen C- oder C++-Programmen heraus. *Scripts, Examples* stellt einige Beispiele für die verschiedenen Programmieraufgaben bereit.

   Mit einem Klick auf *Change* lässt sich außerdem das Installationsverzeichnis ändern; eine gute Wahl ist etwa *C:\MySQL*.

4. *Install* – wenn Sie Ihre Auswahl bestätigen, wird nun die eigentliche Installation durchgeführt.
5. *Signup* – zum Schluss können Sie sich (freiwillig) für die Website *MySQL.com* registrieren. Damit haben Sie Zugriff auf ein gut besuchtes Benutzerforum, in dem Sie sich mit anderen MySQL-Benutzern austauschen und vielleicht Hilfe bei Problemen erhalten können.

Nach der eigentlichen Installation werden Sie gefragt, ob Sie den MySQL-Server sofort konfigurieren möchten. Falls auf Ihrem Rechner garantiert keine alte MySQL-Version installiert war, können Sie dies bedenkenlos bestätigen. Andernfalls sollten Sie die Option zunächst abwählen, um eventuell vorhandene alte Serverdienste zu entfernen.[5]



Windows-Dienste wie der MySQL-Server lassen sich mit Bordmitteln des Betriebssystems nur schwer entfernen – man muss dazu die Windows-Registry manipulieren. Glücklicherweise ist das MySQL-Serverprogramm selbst mit dieser Fähigkeit ausgestattet. Falls Sie einen alten Serverdienst deinstallieren möchten, müssen Sie zuerst in das *bin*-Verzeichnis der neuen MySQL-Installation wechseln – hier die Eingabe für das Beispiel *C:\MySQL\bin*:

```
C:\> cd mysql\bin
```

Anschließend können Sie Folgendes eingeben, um den Dienst zu entfernen:

```
C:\MySQL\bin> mysqld --remove
```

Falls der Dienst nicht *MySQL* heißt, müssen Sie seinen Namen explizit angeben, zum Beispiel:

```
C:\MySQL\bin> mysqld --remove MySQL5
```

5 Das Konfigurationsprogramm enthält zwar eine Funktion, die das automatisch erledigen soll, aber sie funktioniert nicht mit jeder alten Installation.

Falls Sie die automatische Konfiguration abgelehnt haben, können Sie die *MySQL Server Instance Configuration* auch nachträglich starten. Wählen Sie dazu *[Alle] Programme → MySQL → MySQL Server 5.0 → MySQL Server Instance Config Wizard* aus dem Startmenü.

Der Konfigurationsdialog enthält folgende Einzelseiten:

1. *Welcome*: Kurze Information über die Installation. Drücken Sie hier und auf allen folgenden Bildschirmen *Next*, um weiterzublättern.
2. Falls MySQL bereits installiert war, können Sie sich zwischen der Neukonfiguration des vorhandenen Servers (*Reconfigure Instance*) oder seiner Deinstallation (*Remove Instance*, siehe oben) entscheiden.
3. Wählen Sie *Detailed Configuration*, um alle hier beschriebenen Einstellungen manuell vornehmen zu können, oder *Standard Configuration*, wenn Sie nur wenige Optionen anpassen möchten.
4. *Server Type*: Entscheiden Sie sich zwischen einem Arbeitsplatzrechner, auf dem MySQL nur nebenher läuft (*Developer Machine*), einem Serverrechner, auf dem MySQL neben anderen Serverdiensten ausgeführt wird (*Server Machine*), oder einem exklusiv für die Datenbank reservierten Serverrechner (*Dedicated MySQL Server Machine*). Zum Durcharbeiten dieses Buchs ist *Developer Machine* in der Regel die richtige Wahl.
5. *Database Usage*: Entscheiden Sie sich zwischen Transaktionsoptimierung und Performance: *Multifunctional Database* bietet eine ausgewogene Unterstützung für die beiden Haupt-Tabellentypen MyISAM und InnoDB. *Transactional Database* optimiert MySQL für die transaktionsfähigen InnoDB-Tabellen (MyISAM bleibt allerdings möglich). *Non-Transactional Database Only* unterstützt ausschließlich MyISAM-Tabellen, bietet aber die beste Performance.
6. *InnoDB Datafile*: Hier werden Datenträger und Verzeichnis für die Transaktionsdatenbanken gewählt. Tendenziell sollten Sie die Festplatte mit dem meisten freien Platz wählen.
7. *Number of concurrent connections*: Stellen Sie ein, wie viele Verbindungen Ihr Datenbankserver etwa gleichzeitig verarbeiten wird. Für einfache Webanwendungen genügt in der Regel die Option *Decision Support (DSS)/OLAP*, die für 20 gleichzeitige Verbindungen ausgelegt ist. Für gut besuchte E-Commerce-Sites und andere transaktionsorientierte Anwendungen sollten Sie dagegen *Online Transaction Processing (OLTP)* mit Unterstützung für 500 Verbindungen wählen. Unter *Manual Setting* können Sie auch selbst einen Wert eingeben; für einen Entwicklungs- und Testrechner genügt sicherlich eine Vorgabe von fünf bis zehn Verbindungen.
8. *Enable TCP/IP Networking*: Bestimmt, ob der MySQL-Server über das Netzwerk (beziehungsweise Internet) kommunizieren kann oder nur mit lokalen Programmen. In den meisten Fällen wird die Netzwerkunterstützung aktiviert,

selbst wenn Webserver, Webanwendung und Datenbank auf demselben Rechner betrieben werden. Der TCP-Port 3306 ist für MySQL-Server üblich und wird von den meisten Clientprogrammen und Programmierschnittstellen als Standard vorausgesetzt. Da Verbindungen auf lokalen Ports zwischen 0 und 1023 auf Unix-Systemen nur durch `root` geöffnet werden dürfen, hat dies den Sicherheitsvorteil, dass MySQL unter einer weniger privilegierten Benutzer-ID ausgeführt werden kann. Unter Windows spielt dieser Aspekt allerdings keine Rolle.

Enable Strict Mode auf derselben Registerkarte sollte üblicherweise aktiviert bleiben. Durch diese Option werden einige MySQL-Eigenheiten zugunsten des ANSI-SQL-Standards aufgegeben.

9. *Default Character Set*: Hier wird der Standardzeichensatz ausgewählt. Alle in Ihren Datenbanken gespeicherten Textdaten verwenden automatisch den hier gewählten Zeichensatz, solange Sie nicht ausdrücklich einen anderen angeben. Für Deutsch und andere mitteleuropäische Sprachen, die die lateinische Schrift verwenden, ist *Standard Character Set (iso-latin-1)* die beste Wahl. Wählen Sie dagegen *Best Support For Multilingualism (UTF-8)*, wenn Sie Daten in vielen verschiedenen Sprachen mit unterschiedlichen Zeichensätzen verwenden. Sollten Sie dagegen vornehmlich eine bestimmte nicht europäische beziehungsweise nicht lateinisch geschriebene Sprache einsetzen, können Sie den entsprechenden Zeichensatz unter *Manual Selected Default Character Set/Collation* auswählen. Die *Collation* ist übrigens die sprachabhängige Sortierreihenfolge, die auch Umlaute oder andere diakritische Zeichen berücksichtigt (siehe Kapitel 5).
10. *Windows Options*: Hier geht es darum, MySQL als Windows-Dienst einzurichten (*Install As Windows Service*), der beim Hochfahren des Systems automatisch gestartet wird (*Launch the MySQL Server automatically*) und permanent im Hintergrund aktiv ist. Der Dienstname sollte *MySQL* lauten, solange Sie nicht mehrere MySQL-Dienste parallel betreiben.

    Der Betrieb von MySQL als Dienst ist meist die passende Option: Die Datenbank steht so immer zur Verfügung, ohne allzu viele Ressourcen zu verbrauchen – selbst die Performance leistungshungriger 3-D-Action-Spiele wird nicht wesentlich beeinträchtigt, wenn Sie Apache und MySQL als automatisch gestartete Dienste betreiben.[6]

    Die Option *Include Bin Directory in Windows PATH* ist ebenfalls nützlich: Sie fügt *<MySQL-Verzeichnis>\bin* der Umgebungsvariablen *PATH* hinzu, so dass Sie die im Laufe dieses Buchs vorgestellten MySQL-Dienstprogramme aus jedem beliebigen Arbeitsverzeichnis heraus aufrufen können.

6 Das gilt natürlich nicht mehr, wenn diese Serverdienste über das Netzwerk stark beansprucht werden, sondern nur für einen weitgehend passiven »Standby-Betrieb« auf einem Entwicklerrechner.

11. *Security Options*: Hier werden grundlegende Anmelde- und Sicherheitseinstellungen vorgenommen. Zunächst sollten Sie *Modify Security Settings* ankreuzen. Geben Sie unter *New root password* ein sicheres Passwort[7] für den MySQL-Administrator *root* ein und wiederholen Sie es bei der Option *Confirm*.
    Die Option *Enable root access from remote machines* sollten Sie, wenn möglich, deaktivieren. Auf diese Weise kann *root* nicht über das Netzwerk zugreifen; das verhindert automatisierte Internetangriffe, die von Zeit zu Zeit stattfinden. In aller Regel sollten Sie aus Sicherheitsgründen auch *Create Anonymous Account* deaktivieren – das verhindert Zugriffe ohne Benutzername und Passwort.
12. Schließlich können Sie die gewählten Einstellungen mittels *Execute* annehmen. Wenn dabei einer der Punkte scheitert, müssen Sie zurückblättern und entsprechende Änderungen vornehmen. Am häufigsten scheitert die Konfiguration daran, dass bereits ein MySQL-Dienst installiert ist (siehe oben).

MySQL-Konfigurationsdateien?

Anders als viele andere Open Source-Programme aus dem Unix-Umfeld funktioniert MySQL auch ohne die Existenz einer Konfigurationsdatei. Dennoch besteht natürlich die Möglichkeit, eine solche Datei einzurichten. Zum Lieferumfang von MySQL gehören mehrere Vorlagen mit typischen Einstellungen für kleine, mittlere und große Datenbanken. Sie befinden sich im Verzeichnis *support-files* Ihrer MySQL-Installation (Unix) beziehungsweise direkt im MySQL-Verzeichnis (Windows). Die Dateiendung ist unter Unix *.cnf* und unter Windows *.ini*. Der jeweilige Dateiname (*my-small.cnf*/*my-small.ini* bis *my-innodb-heavy-4G.cnf*/*my-innodb-heavy-4G.ini*) weist auf die Datenbankgröße hin, für die sie zu empfehlen sind. Kommentare im Text der Dateien selbst verraten Genaueres.

Wenn Sie eine der Dateien als Grundlage Ihrer MySQL-Installation einsetzen möchten, müssen Sie sie auf Unix-Systemen in *my.cnf* umbenennen und ins Verzeichnis */etc* kopieren. Unter Windows muss die Datei dagegen *my.ini* heißen und ins Systemverzeichnis (zum Beispiel *C:\Windows*) kopiert werden. Danach müssen Sie MySQL neu starten. Dies geschieht auf Windows-Rechnern mithilfe der Diensteverwaltung (*Start* → *Systemsteuerung* → *Verwaltung* → *Dienste*); auf Unix-Systemen kommt je nach Installationseinstellungen und -verzeichnissen (siehe oben) zum Beispiel folgende Eingabe dafür in Frage:

```
> /etc/init.d/mysql5 restart
```

Diverse Beispiele für Einstellungen, die Sie manuell in Konfigurationsdateien einfügen oder ändern können, werden in Kapitel 9 gezeigt.

7 Gute Passwörter bestehen aus einer Mischung aus Groß- und Kleinbuchstaben sowie Ziffern und kommen in keinem Wörterbuch vor.

MySQL testen

Ob der Datenbankserver korrekt funktioniert, erfahren Sie am einfachsten, indem Sie mit einem Client darauf zugreifen und SQL-Operationen ausführen. Der mitgelieferte Kommandozeilenclient mysql genügt zu diesem Zweck. Er befindet sich im *bin*-Verzeichnis Ihrer MySQL-Installation (zum Beispiel */usr/local/mysql/bin* beziehungsweise *C:\MySQL\MySQL Server 5.0\bin*); wenn Sie der obigen Installationsanleitung gefolgt sind, befindet er sich in beiden Systemen bereits in Ihrem `PATH`.

Unter Windows – wo Sie bereits bei der Installation ein `root`-Passwort festgelegt haben – können Sie Folgendes eingeben:

```
> mysql -u root -p
```

Anschließend werden Sie zur Passworteingabe aufgefordert. Alternativ können Sie auch *Start → [Alle] Programme → MySQL → MySQL Server 5.0 → MySQL Command Line Client* aufrufen und in dem dadurch neu geöffneten Konsolenfenster sofort das `root`-Passwort eingeben.

Auf einem Unix-System genügt zunächst folgende Eingabe:

```
$ mysql
```

Hier sollten Sie umgehend ein Passwort für den MySQL-Benutzer `root` anlegen. Dies funktioniert mit dem – einfallslosen und recht unsicheren – Passwort *geheim01* folgendermaßen:

```
mysql> SET PASSWORD FOR root@localhost=PASSWORD('geheim01');
```

Anschließend müssen Sie alle Einträge ohne Benutzernamen und/oder ohne Passwort aus der Verwaltungstabelle `user` löschen:

```
mysql> DELETE FROM mysql.user WHERE user="" OR PASSWORD="";
```

Wenn Änderungen an Benutzerrechten bereits während der Clientsitzung aktiv werden sollen, brauchen Sie anschließend noch folgende Anweisung:

```
mysql> FLUSH PRIVILEGES;
```

Beim nächsten Mal funktioniert der Start des Clients dann wie unter Windows.



Sehr wahrscheinlich ist der Zeichensatz Ihres Konsolenfensters inkompatibel zur MySQL-Standardeinstellung `iso-latin-1`. Näheres dazu erfahren Sie in Kapitel 4, aber im Moment sollten Sie den innerhalb des Clients verwendeten Zeichensatz anpassen, bevor Sie Datenbanken oder Tabellen mit fehlerhaften Umlauten erzeugen. Die meisten modernen Linux-Terminals verwenden `utf-8`, also geben Sie hier Folgendes ein:

```
mysql> SET NAMES utf8;
```

Die Windows-Eingabeaufforderung verwendet dagegen als DOS-Erbe einen ganz eigenen Zeichensatz, der auch zur grafischen Oberfläche von Windows inkompatibel ist. Geben Sie hier deshalb folgende Anweisung ein:

```
mysql> SET NAMES cp850;
```

Als Nächstes wird eine Datenbank mit einer kleinen Tabelle angelegt. Die Eingabe in `mysql` ist etwas gewöhnungsbedürftig: Befehle werden nicht durch Enter abgeschlossen, sondern erst durch ein Semikolon. Dies gibt Ihnen Gelegenheit, längere Anweisungen in mehrere Zeilen zu strukturieren; in Nachfolgezeilen wird statt `mysql>` ein Pfeil (`->`) angezeigt. Geben Sie zunächst Folgendes ein, um die Datenbank zu erstellen:

```
mysql> CREATE DATABASE geotest;
Query OK, 1 row affected (0.18 sec)
```

Beachten Sie, dass bei SQL-Abfragen nicht zwischen Groß- und Kleinschreibung unterschieden wird; traditionell – und auch hier im Buch – werden sie komplett in Großbuchstaben geschrieben. Bei den Namen von Datenbanken, Tabellen, Feldern und anderen selbst gewählten Bezeichnern ist die Unterscheidung zwischen Groß- und Kleinschreibung dagegen plattformabhängig: Unter Unix wird sie beachtet, unter Windows nicht – genau wie bei Dateien auf diesen Systemen. Konsequenterweise sollten Sie sich immer an eine einheitliche Schreibweise halten; in diesem Buch werden deshalb alle Bezeichner kleingeschrieben.

Nun wird die neu erstellte Datenbank als Standard ausgewählt:

```
mysql> USE geotest
Database changed
```

In der Datenbank soll eine einzelne Tabelle namens *laender* erstellt werden. Sie enthält eine Liste von Ländern mit deren Hauptstädten und den auswählbaren Kontinenten. Geben Sie dazu Folgendes ein:

```
mysql> CREATE TABLE laender (
    ->   land VARCHAR(30),
    ->   hauptstadt VARCHAR(30),
    ->   kontinent ENUM('Afrika', 'Asien', 'Australien', 'Europa',
    ->  'Nordamerika', 'Südamerika')
    -> );
Query OK, 0 rows affected (1.10 sec)
```

Der Feldtyp `VARCHAR(`*n*`)` speichert Text bis zur maximalen Länge von *n* Zeichen. `ENUM` ist dagegen eine Auflistung fester Werte, die relativ wenig Speicherplatz benötigt.

Als Nächstes sollten einige Werte hinzugefügt werden:

```
mysql> INSERT INTO laender VALUES
    ->    ('Frankreich', 'Paris', 'Europa'),
    ->    ('Italien', 'Rom', 'Europa'),
    ->    ('Japan', 'Tokio', 'Asien'),
    ->    ('Marokko', 'Rabat', 'Afrika'),
```

```
    ->    ('Australien', 'Canberra', 'Australien'),
    ->    ('Kanada', 'Ottawa', 'Nordamerika'),
    ->    ('Argentinien', 'Buenos Aires', 'Südamerika');
Query OK, 7 rows affected (0.04 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

Wie Sie sehen, können Sie beliebig viele geklammerte Wertegruppen durch Kommata trennen; erst das übliche Semikolon schließt die Eingabe ab.

Folgendermaßen können Sie sich die gesamte Tabelle alphabetisch nach Ländern geordnet anzeigen lassen:

```
mysql> SELECT * FROM laender ORDER BY land ASC;
+-------------+--------------+-------------+
| land        | hauptstadt   | kontinent   |
+-------------+--------------+-------------+
| Argentinien | Buenos Aires | Südamerika  |
| Australien  | Canberra     | Australien  |
| Frankreich  | Paris        | Europa      |
| Italien     | Rom          | Europa      |
| Japan       | Tokio        | Asien       |
| Kanada      | Ottawa       | Nordamerika |
| Marokko     | Rabat        | Afrika      |
+-------------+--------------+-------------+
7 rows in set (0.00 sec)
```

Typischer sind SELECT-Abfragen, in denen eine WHERE-Option (*Klausel*) nur bestimmte Datensätze auswählt. Das folgende Beispiel zeigt nur die Datensätze mit dem Kontinent »Europa« an:

```
mysql> SELECT * FROM laender WHERE kontinent='Europa';
+------------+------------+-----------+
| land       | hauptstadt | kontinent |
+------------+------------+-----------+
| Frankreich | Paris      | Europa    |
| Italien    | Rom        | Europa    |
+------------+------------+-----------+
2 rows in set (0.05 sec)
```

Nach diesen einfachen Tests können Sie davon ausgehen, dass Ihr MySQL-Server ordnungsgemäß funktioniert. Deshalb können Sie sich nun abmelden:

```
mysql> exit
```

PHP installieren

Der letzte Bestandteil eines LAMP- oder WAMP-Systems ist die Programmiersprache PHP (ein rekursives Akronym für **P**HP: **H**ypertext **P**reprocessor). Die Sprache wurde seit 1995 von *Rasmus Lerdorf* entwickelt. Anfangs handelte es sich um eine einfache Sammlung von Makros für automatisierte Websites (*Personal Homepage Tools*), sie wurde aber schnell zu einer vollwertigen Programmiersprache mit dem Schwerpunkt Webanwendungen ausgebaut. In der aktuellen Version 5 wurde vor

allem die Objektorientierung verbessert; außerdem konnte die Performance durch den neuen Interpreter-Kern *Zend Engine II* gesteigert werden.

PHP ist für allerlei Unix-Varianten, für Windows sowie für einige andere Systeme verfügbar. In diesem Abschnitt wird auf die Installation unter Linux und Windows eingegangen. Sie können PHP entweder über die klassische CGI-Schnittstelle des Webservers betreiben oder als integriertes Webserver-Modul. Im Folgenden werden beide Möglichkeiten beschrieben (die Modulvariante am Beispiel Apache 2). Für eine Website im praktischen Einsatz ist der Betrieb als Modul dringend zu empfehlen, da CGI umständlich und langsam ist.

Installation unter Linux

Es gibt keine offiziellen PHP-Binaries für Linux und andere Unix-Systeme. Sofern PHP nicht Bestandteil Ihrer Systemdistribution ist oder nur in einer veralteten Version mitgeliefert wurde, müssen Sie das Quellcode-Paket installieren. Entpacken Sie es zunächst in ein beliebiges Verzeichnis:

```
# tar -xzvf php-5.2.2.tar.gz
```

Die restlichen Schritte sollten Sie, wie üblich, als `root` ausführen. Genau wie bei Apache oder bei der MySQL-Quellcode-Distribution müssen die drei Befehle `configure`, `make` und `make install` ausgeführt werden.

Das Skript *configure* bietet zahlreiche Optionen. Wenn Sie alle nachlesen wollen, können Sie es mit der Option `--help` aufrufen (siehe oben die MySQL-Installationsanleitung). Hier nur die vier wichtigsten Optionen:

- Mit der Option `--prefix=<Pfad>` können Sie das Verzeichnis der PHP-Installation wählen, üblicherweise */usr/local/php* oder */usr/local/php5*.
- Wenn PHP als Apache 2-Modul installiert werden soll, muss die Option `--with-apxs2` verwendet werden. Sie benötigt den Pfad zu dem Skript *apxs* (Apache Extension Tool) im `bin`-Verzeichnis Ihrer Apache-Installation, zum Beispiel `--with-apxs2=/usr/local/apache2/bin/apxs`.
- Für die Programmierung MySQL-basierter Webanwendungen sind die MySQL-Schnittstellen wichtig. Wie Sie im Verlauf dieses Buchs erfahren werden, gibt es drei verschiedene: *mysql*, *mysqli* sowie *PHP Data Objects (PDO)* mit MySQL-Treiber – bestehend aus den beiden Erweiterungen *pdo* und *pdo_mysql*. Die klassische mysql-Schnittstelle wird mittels `--with-mysql=<MySQL-Pfad>` hinzugefügt, zum Beispiel: `--with-mysql=/usr/local/mysql/`.
- Um die moderneren Schnittstellen *mysqli* und *PDO* einzurichten, wird die Option `--with-mysqli` beziehungsweise `--with-pdo --with-pdo_mysql` verwendet. Als Argument benötigen sie den Pfad des Tools *mysql_config* im *bin*-Verzeichnis der MySQL-Installation, also beispielsweise `--with-mysqli=/usr/local/mysql/bin/mysql_config` oder `--with-pdo  --with-pdo_mysql=/usr/local/mysql/bin/mysql_config`.

Das folgende Beispiel konfiguriert PHP mit allen MySQL-Schnittstellen als Apache 2-Modul zur Installation im Verzeichnis */usr/local/php5*:

```
# ./configure --prefix=/usr/local/php5 \
--with-apxs2=/usr/local/apache2/bin/apxs \
--with-mysql=/usr/local/mysql \
--with-mysqli=/usr/local/mysql/bin/mysql_config \
--with-pdo --with-pdo_mysql=/usr/local/mysql/bin/mysql_config
```

Das nächste Beispiel bereitet PHP dagegen zur Installation als externe CGI-Sprache ins Verzeichnis */usr/local/php* vor, wobei nur die klassische *mysql*-Schnittstelle eingerichtet wird:

```
# ./configure --prefix=/usr/local/php --with-mysql=/usr/local/mysql
```

Wie üblich werden nach der Konfiguration die Standardbefehle zur Kompilierung und Installation aufgerufen:

```
# make
# make install
```

Wenn Sie PHP als Apache-Modul installiert haben, hat `apxs` automatisch folgende Zeile zur Konfigurationsdatei *httpd.conf* hinzugefügt:

```
LoadModule php5_module modules/libphp5.so
```

Zusätzlich müssen Sie manuell dafür sorgen, dass Dateien mit der Endung *.php* als PHP 5-Skripten erkannt werden. Fügen Sie dazu unter den vorhandenen `AddHandler`-Einträgen folgende Direktive zur *httpd.conf* hinzu:

```
AddHandler php5-script .php
```

Wenn Sie möchten, können Sie – durch Leerzeichen getrennt – auch die veralteten Endungen `.php3`, `.php4` und `.phtml` hinzufügen.

Haben Sie PHP nicht als Modul installiert und möchten Ihre Skripten stattdessen als CGI-Anwendungen ausführen, müssen Sie (an den jeweils passenden Stellen der Konfigurationsdatei) folgende Zeilen einfügen:

```
Alias /php/ /usr/local/php/bin/
Action application/x-httpd-php "/php/php"
```

Natürlich muss der PHP-Pfad in der `Alias`-Direktive an Ihr System angepasst werden.

Beachten Sie, dass Sie Apache nach der PHP-Installation meist beenden und wieder starten müssen; ein einfacher Neustart genügt nicht. Anschließend können Sie PHP testen. Speichern Sie die drei folgenden Zeilen in einer Datei mit der Endung *.php* – zum Beispiel *info.php* – in Ihrem Website-Verzeichnis (bei einer Apache-Standardinstallation */usr/local/apache2/htdocs*):

```
<?php
   phpinfo();
?>
```

Wenn Sie dieses Dokument nun über *http://localhost/info.php* in einem Browser öffnen, wird eine umfangreiche Übersicht über die aktuelle PHP- und Apache-Konfiguration angezeigt. Beachten Sie, dass Sie PHP-Dokumente nicht als lokale Dateien im Browser öffnen können – es ist ein Webserver erforderlich, der den PHP-Code ausführt.

Installation unter Windows

Für Windows wird PHP in zwei Varianten angeboten: als Installer und als ZIP-Archivdatei. Der Installer konnte bis vor Kurzem nur die CGI-Variante installieren. Inzwischen enthält er auch experimentelle Unterstützung für die automatische Installation als Webserver-Modul, aber diese funktioniert noch nicht zuverlässig. Deshalb beschränkt sich die folgende Anleitung auf die ZIP-Datei (zurzeit *php-5.2.2-Win32.zip*):

1. Entpacken Sie die Datei in das gewünschte PHP-Verzeichnis. Die nachfolgenden Anweisungen gehen von *C:\php* aus, aber Sie können natürlich auch ein anderes Verzeichnis wählen.
2. Kopieren Sie die Datei *php.ini-dist*, die Vorlage für die PHP-Konfigurationsdatei, in Ihr Systemverzeichnis (zum Beispiel *C:\Windows* oder *C:\WinNT*) und benennen Sie sie dort in *php.ini* um.
3. Nehmen Sie in der neuen *php.ini*-Datei die folgenden Änderungen vor (diese Zeilen existieren bereits, Sie müssen nur die konkreten Verzeichnisse eintragen):

   ```
   doc_root = C:\Apache2\htdocs  ; Ihre Apache-DocumentRoot
   extension_dir = C:\php\ext    ; ext in Ihrem PHP-Verzeichnis
   ```

   Als Nächstes werden die gewünschten PHP-Erweiterungen (*mysql*, *mysqli* und/oder *pdo_mysql*) aktiviert:

   ```
   extension=php_mysql.dll
   extension=php_mysqli.dll
   extension=php_pdo.dll
   extension=php_pdo_mysql.dll
   ```

   Manche der gewünschten Zeilen sind möglicherweise bereits vorhanden, sind aber durch ein führendes Semikolon (`;`) auskommentiert, das Sie in diesem Fall nur zu entfernen brauchen. Andere müssen Sie möglicherweise manuell hinzufügen.

   Daneben können Sie nach Belieben weitere Erweiterungen aktivieren. Das Web-Interface phpMyAdmin (siehe unten) benötigt beispielsweise die Erweiterung *mbstring.dll*; löschen Sie hier ebenfalls das Kommentarzeichen. Im nächsten Abschnitt erhalten Sie weitere Informationen über die *php.ini*-Optionen.
4. Kopieren Sie die Datei *php5ts.dll* aus dem PHP-Verzeichnis nach *<Windows-Verzeichnis>\System32*.

5. Fügen Sie das PHP-Installationsverzeichnis zur Umgebungsvariablen *PATH* hinzu. Klicken Sie dazu mit der rechten Maustaste auf das Desktop-Symbol *Arbeitsplatz* und wählen Sie *Eigenschaften* aus dem Kontextmenü. Klicken Sie unter Windows 2000 und XP auf der Registerkarte *Erweitert* die Schaltfläche *Umgebungsvariablen* an; Windows NT 4.0 besitzt eine separate Registerkarte namens *Umgebung*. Doppelklicken Sie im entsprechenden Dialog auf die Variable *PATH*. Hängen Sie an den bestehenden Wert ein Semikolon und den vollständigen Pfad Ihres PHP-Verzeichnisses an, zum Beispiel *C:\php*. Beachten Sie, dass die Änderung nach einem Klick auf *OK* nur durch einen vollständigen Systemneustart wirksam wird.



Auf einigen Systemen werden die MySQL-Erweiterungen auch nach diesem Schritt noch nicht ordnungsgemäß geladen. Kopieren Sie in diesem Fall auch die Datei *libmysql.dll* aus dem PHP-Verzeichnis nach *<Windows-Verzeichnis>\System32*. Manchmal entsteht das Problem gerade dadurch, dass sich an dieser Stelle eine inkompatible Version dieser Datei aus einer früheren PHP-Installation befindet.

6. Führen Sie je nach PHP-Betriebsart die entsprechenden Änderungen an der Apache-Konfigurationsdatei *httpd.conf* durch. Für die Modulvariante müssen Sie folgende Zeilen einfügen (am besten jeweils an den Stellen, an denen bereits identische Direktiven mit anderen Werten stehen):

   ```
   LoadModule php5_module "C:/php/php5apache2.dll"
   AddHandler php5-script .php
   ```

   Natürlich müssen Sie die Pfadangabe in der `LoadModule`-Direktive an Ihre PHP-Installation anpassen.

   Wenn Sie PHP über CGI verwenden möchten, müssen die folgenden Zeilen hinzugefügt werden:

   ```
   ScriptAlias /php/ "c:/php/"
   AddType application/x-httpd-php .php
   Action application/x-httpd-php "/php/php.exe"
   ```

Nachdem Sie alle Schritte abgeschlossen haben, müssen Sie Apache beenden und wieder starten. Anschließend sollten Sie die ordnungsgemäße Funktionalität von PHP testen. Dazu können Sie eine PHP-Datei mit folgendem Inhalt schreiben:

```
<?php
   phpinfo();
?>
```

Speichern Sie die Datei innerhalb Ihres Apache-Website-Verzeichnisses, zum Beispiel *C:\Programme\Apache Software Foundation\Apache2.2\htdocs* (PHP-Dokumente können nicht als lokale Dateien im Browser geöffnet werden – es ist ein Webserver erforderlich, der den PHP-Code ausführt). Wenn Sie die Datei nun in einem Browser öffnen, wird eine umfangreiche Übersicht über die aktuelle PHP- und Apache-Konfiguration angezeigt.

Die Konfigurationsdatei php.ini

Wie bereits erwähnt, können Sie das Verhalten von PHP mithilfe der PHP-Konfigurationsdatei *php.ini* anpassen. Ihre Syntax wurde den klassischen Windows-INI-Dateien angepasst, daraus resultieren folgende Regeln:

- Abschnitte werden durch Schlüsselwörter in eckigen Klammern eingeleitet, zum Beispiel `[PHP]`.
- Die Direktiven haben das Format *Name* = *Wert*, zum Beispiel:

  ```
  doc_root = /usr/local/apache2/htdocs
  ```

  Beachten Sie in diesem Zusammenhang, dass Windows – anders als in der Apache-Konfiguration – den plattformtypischen Backslash (`\`) und nicht den Unix-Slash (`/`) als Pfadtrennzeichen verwendet.
- Der Rest einer Zeile hinter einem Semikolon ist ein Kommentar. Beispiele:

  ```
  ; doc_root: Die DocumentRoot der Website
  doc_root = "C:\Programme\Apache Software Foundation\Apache2.2\htdocs"  ; Win32
  ```

In Tabelle 2-3 sehen Sie die wichtigsten Direktiven auf einen Blick.

Tabelle 2-3: Wichtige Einstellungen für die PHP-Konfigurationsdatei php.ini

Direktive	Vorgabewert	Aufgabe
`short_open_tag = On|Off`	`On`	`On` erlaubt `<? ... ?>` als Abkürzung für `<?php ... ?>`
`asp_tags = On|Off`	`Off`	`On` erlaubt `<% ... %>` (ASP-Syntax) statt `<?php ... ?>`
`max_execution_time = Sekunden`	`30`	maximale Ausführungsdauer von PHP-Skripten
`precision = "n"`	`"14"`	Fließkommazahlen mit *n* Stellen Genauigkeit
`expose_php = On|Off`	`On`	`On` fügt die PHP-Version zur Serveridentifikation hinzu
`memory_limit = "...M|K|B"`	`"8M" (8 Megabyte)`	maximaler Arbeitsspeicher für PHP-Skripten (verfügbar, wenn PHP mit `--enable-memory-limit` kompiliert wurde)
`track-vars = "On"|"Off"`	`"On"`	Spezialvariablen `$_ENV`, `$_GET`, `$_POST`, `$_COOKIE` und `$_SERVER` mit HTTP-Daten und Umgebungsvariablen ein/aus
`arg_separator.output = "STRING"`	`"&"`	Trennzeichen, das PHP in selbst erzeugte Query-Strings einfügt
`arg_separator.input = "STRING"`	`"&"`	Trennzeichen, die PHP beim Auswerten von Query-Strings akzeptiert (mehrere möglich, zum Beispiel `"&;"`)
`variables_order = "STRING"`	`"EGPCS"`	Variablenreihenfolge für `register_globals`: `E` = Umgebungsvariablen, `G` = GET-Felder, `P` = POST-Felder, `C` = Cookies, `S` = Servervariablen

Tabelle 2-3: Wichtige Einstellungen für die PHP-Konfigurationsdatei php.ini (Fortsetzung)

Direktive	Vorgabewert	Aufgabe
`register_argc_argv = "On"|"Off"`	`"On"`	GET-Variablen und Kommandozeilenargumente als `$argc` (Anzahl) und Array `$argv` (Werte) im C-Stil
`post_max_size = "...M|K|B"`	`"8M"`	maximale Größe für POST-Daten
`auto_prepend_file = "DATEIPFAD"`	`""`	automatisch am Kopf jeder Datei importierte Include-Datei
`auto_append_file = "DATEIPFAD"`	`""`	automatisch am Fuß jeder Datei importierte Include-Datei
`default_mimetype = "TYPE/SUBTYPE"`	`"text/html"`	Standardwert für `Content-Type`
`default_charset = "ZEICHENSATZ"`	`"iso-8859-1"`	Standardzeichensatz für `Content-Type`
`always_populate_raw_post_ data = "0"|"1"`	`"0"`	`"1"` erzeugt stets `$HTTP_RAW_POST_DATA` (String mit nicht getrennten POST-Daten)
`include_path = "PFAD1[:PFAD2:...]"`	Umgebungsvariable `PHP_INCLUDE_PATH`	Verzeichnisse, in denen PHP automatisch nach Include-Dateien sucht
`doc_root = "PFAD1[:PFAD2:...]"`	`""`	Stammverzeichnis für PHP-Skripten (in der Regel Apache-`DocumentRoot`)
`user_dir = "VERZEICHNIS"`	`NULL`	Benutzerverzeichnis mit PHP-Dateien (`/home/USER/...`)[a]
`extension_dir = "PFAD"`	Umgebungsvariable `PHP_EXTENSION_DIR`	Verzeichnis für PHP-Extensions
`extension = "PFAD"`	`[diverse]`	angegebene Erweiterung laden (bei relativer Pfadangabe aus `extension_dir`)
`file_uploads = "0"|"1"`	`"1"`	`"1"` erlaubt Datei-Uploads
`upload_tmp_dir = "PFAD"`	`NULL (Systemvorgabe)`	temporäres Upload-Verzeichnis
`upload_max_filesize = "...M|K|B"`	`"2M"`	maximale Größe für Upload-Dateien

[a] Sinnvoll im Zusammenhang mit der Apache-Direktive `UserDir`, die benutzerspezifische Websites wie *http://www.test.local/~username* zulässt. Siehe *http://httpd.apache.org/docs-2.0/mod/mod_userdir.html*.

Den MySQL-Zugriff konfigurieren und testen

Die Kommunikation zwischen PHP und dem MySQL-Server erfolgt aufgrund einer Benutzeranmeldung; Benutzername und Passwort sind Parameter der PHP-Funktionen für die Datenbankverbindung. Einen solchen Benutzer müssen Sie zunächst einmal erstellen – aus Sicherheitsgründen sollten Sie nicht das MySQL-Administratorkonto root einsetzen.

Angenommen, Sie möchten einen Benutzer namens *phpuser* mit dem Passwort *PHP4ever* einrichten, der alle Operationen an der Testdatenbank *geotest* vorneh-

men darf. Öffnen Sie dazu den Kommandozeilenclient mysql (siehe oben) und geben Sie folgende Befehle ein:

```
mysql> CREATE USER phpuser@localhost IDENTIFIED BY "PHP4ever";
mysql> GRANT ALL PRIVILEGES ON geotest.* TO phpuser@localhost;
mysql> FLUSH PRIVILEGES;
```

In der Praxis sollten Sie die Rechte von PHP-MySQL-Benutzern erheblich einschränken. In der Regel genügt beispielsweise der Zugriff auf einzelne Datenbanktabellen; auch `ALL PRIVILEGES` (alle Rechte) muss nicht sein. Mehr über das MySQL-Berechtigungsmanagement erfahren Sie in Kapitel 9, *MySQL-Administration*.

Wenn Web- und MySQL-Server auf unterschiedlichen Rechnern laufen, müssen Sie `@localhost` durch den Domainnamen oder die IP-Adresse des gewünschten Rechners ersetzen. Sie können den Zusatz auch ganz weglassen, um den Zugriff von beliebigen Rechnern aus zu gestatten; das ist allerdings weniger sicher und daher nicht zu empfehlen.



Auf keinen Fall dürfen Sie den hier als Beispiel verwendeten Benutzernamen *phpuser* zusammen mit dem Passwort *PHP4ever* verwenden! Die Informationen in diesem Buch sind öffentlich verfügbar, so dass potenzielle Angreifer auf die Idee kommen könnten, diese Werte auf Verdacht gegen fremde MySQL-Server auszuprobieren.

Um zu testen, ob Sie unter der neuen Benutzer-ID aus PHP-Skripten auf die MySQL-Datenbank zugreifen können, sollten Sie zunächst eine kleine Testdatenbank einrichten. Das Staaten/Hauptstädte-Beispiel aus dem vorigen Abschnitt reicht dafür völlig aus. Hier wird ein kurzes PHP-Skript vorgestellt, das alle Datensätze ordentlich als HTML-Tabelle ausgibt. Es werden drei Lösungen für die verschiedenen Schnittstellen *mysql*, *mysqli* und *pdo_mysql* erstellt.

Formal betrachtet, sind PHP-Dateien gewöhnliche HTML-Dokumente. Die eigentlichen PHP-Anweisungen befinden sich an beliebigen Stellen darin und werden von einem Block wie diesem umschlossen:

```
<?php

   // hier PHP-Anweisungen

?>
```

Als Erstes muss in dem Skript die Verbindung zum MySQL-Server hergestellt und die Standarddatenbank ausgewählt werden. Mit der klassischen *mysql*-Schnittstelle funktioniert das folgendermaßen:

```
// Datenbankverbindung: localhost, User phpuser, Passwort PHP4ever
$connID = mysql_connect ("localhost", "phpuser", "PHP4ever");
// Datenbank "geotest" auswählen
mysql_select_db ("geotest");
```

Die Sequenz // leitet übrigens einen PHP-Kommentar ein; der Rest der Zeile wird ignoriert. Alternativ können Sie auch # verwenden; zusätzlich wird der mehrzeilige Kommentar /* ... */ im C-Stil unterstützt.

Wenn Sie die Schnittstelle *mysqli* verwenden, benötigen Sie nur eine Zeile. Hier wird ein Verbindungsobjekt erstellt, in dem unter anderem die Standarddatenbank gespeichert ist:

```
$conn = new mysqli("localhost", "phpuser", "PHP4ever", "geotest");
```

Bei *PHP Data Objects* wird ebenfalls ein Objekt erzeugt, in diesem Fall von der Klasse PDO. Es werden drei Argumente benötigt. Das erste ist ein String mit dem Treibernamen, gefolgt von einem Doppelpunkt sowie treiberspezifischen, durch Semikola getrennten Name=Wert-Paaren. Als zweites und drittes Argument werden Username und Passwort angegeben. Für das aktuelle Beispiel sieht das Ganze so aus:

```
$conn = new PDO
     ("mysql:host=localhost;dbname=geotest", "phpuser", "PHP4ever");
```

Die SQL-Abfrage zur Ausgabe aller Länderinformationen, die an den Datenbankserver übergeben werden soll, lautet in jedem Fall:

```
SELECT * FROM laender ORDER BY land ASC
```

Eine Datenbankabfrage wird bei *mysql* mithilfe der Funktion mysql_query() durchgeführt. *mysqli* und *PDO* verwenden dagegen die Methode query() des jeweiligen Verbindungsobjekts. Der entsprechende Code sieht so aus:

```
// mysql
$result = mysql_query ("SELECT * FROM laender ORDER BY land ASC");
// mysqli, PDO
$result = $conn->query ("SELECT * FROM laender ORDER BY land ASC");
```

Anschließend müssen die Ergebniszeilen aus der Datenbank gelesen werden. Dazu dient bei *mysql* die Funktion mysql_fetch_row(), die jeweils den nächsten Datensatz des Abfrageergebnisses liest. Das Ergebnis ist ein Array (eine mehrgliedrige Variable mit nummeriertem Index) mit den einzelnen Feldern des Datensatzes. In PHP existiert die äußerst praktische Möglichkeit, dieses Array über die Funktion list() einer selbst gewählten Gruppe von Variablen zuzuordnen. Übrigens können Sie den Aufruf mysql_fetch_row() zur Bedingung einer Schleife machen. Die äußere Struktur sieht demnach folgendermaßen aus:

```
while (list ($land, $hauptstadt, $kontinent)
      = mysql_fetch_row ($result)) {
   // Werte anzeigen ...
}
```

mysqli tut sinngemäß dasselbe, verwendet allerdings die Methode fetch_row() des query-Objekts:

```
while (list ($land, $hauptstadt, $kontinent)
      = $result->fetch_row()) {
   // Werte anzeigen
}
```

Bei PDO heißt die zuständige Methode einfach `fetch()`; sie hat zusätzliche Fähigkeiten, erledigt die hier geforderte Aufgabe aber genauso:

```
while (list ($land, $hauptstadt, $kontinent) = $result->fetch()) {
   // Werte anzeigen
}
```

An einem Punkt Ihres Skripts, an dem Sie die Datenbankverbindung nicht mehr brauchen, sollten Sie sie schließen. Auch dies funktioniert je nach Schnittstelle unterschiedlich:

```
// mysql
mysql_close();

// mysqli
$conn->close();

// PDO
$conn = null;
```

Beispiel 2-1 enthält das gesamte Skript in *mysql*-Syntax, in Beispiel 2-2 sehen Sie dasselbe Programm in objektorientierter *mysqli*-Syntax und in Beispiel 2-3 noch einmal mit PDO. Abbildung 2-1 zeigt, wie das Ergebnis im Browser aussieht. Mehr Einzelheiten über PHP erfahren Sie in den nachfolgenden Kapiteln.

Beispiel 2-1: laender_mysql.php – eine SQL-Abfrage auslesen und als HTML-Tabelle anzeigen

```
<html>
  <head>
    <title>Einfache MySQL-Abfrage mit PHP, mysqli-Version</title>
  </head>
  <body>
    <h1>Einige L&auml;nder, ihre Hauptst&auml;dte und Kontinente</h1>
    <table border="2" cellpadding="4">
      <tr>
        <th>Land</th>
        <th>Hauptstadt</th>
        <th>Kontinent</th>
      </tr>
<?php

  // Parameter für die Datenbankverbindung
  $host = "localhost";
  $user = "phpuser";
  $pass = "PHP4ever";
  $db   = "geotest";
  // Datenbankverbindung herstellen
  $connID = mysql_connect ($host, $user, $pass);
  // Datenbank auswählen
  mysql_select_db ($db);
  // MySQL-Abfrage
  $result = mysql_query ("SELECT * FROM laender ORDER BY land ASC");
```

Beispiel 2-1: laender_mysql.php – eine SQL-Abfrage auslesen und als HTML-Tabelle anzeigen (Fortsetzung)

```
  // Schleife zum Auslesen und Anzeigen der Ergebnisse
  while (list ($land, $hauptstadt, $kontinent)
         = mysql_fetch_row ($result)) {
    echo "<tr>\n";
    echo "<td>$land</td>\n";
    echo "<td>$hauptstadt</td>\n";
    echo "<td>$kontinent</td>\n";
    echo "</tr>\n";
  }
  // Datenbankverbindung schließen
  mysql_close();

?>
    </table>
  </body>
</html>
```

Beispiel 2-2: laender_mysqli.php – dasselbe Skript in objektorientierter mysqli-Syntax

```
<html>
  <head>
    <title>Einfache MySQL-Abfrage mit PHP, mysqli-Version</title>
  </head>
  <body>
    <h1>Einige L&auml;nder, ihre Hauptst&auml;dte und Kontinente</h1>
    <table border="2" cellpadding="4">
      <tr>
        <th>Land</th>
        <th>Hauptstadt</th>
        <th>Kontinent</th>
      </tr>
<?php

  // Parameter für die Datenbankverbindung
  $host = "localhost";
  $user = "phpuser";
  $pass = "PHP4ever";
  $db   = "geotest";
  // Verbindung herstellen und Datenbank auswählen
  $conn = new mysqli ($host, $user, $pass, $db);
  // MySQL-Abfrage
  $result = $conn->query ("SELECT * FROM laender ORDER BY land ASC");
  // Schleife zum Auslesen und Anzeigen der Ergebnisse
  while (list ($land, $hauptstadt, $kontinent) =
         $result->fetch_row()) {
    echo "<tr>\n";
    echo "<td>$land</td>\n";
    echo "<td>$hauptstadt</td>\n";
    echo "<td>$kontinent</td>\n";
    echo "</tr>\n";
  }
```

Beispiel 2-2: laender_mysqli.php – dasselbe Skript in objektorientierter mysqli-Syntax (Fortsetzung)

```
  // Datenbankverbindung schließen
  $conn->close();

?>
    </table>
  </body>
</html>
```



Abbildung 2-1: Darstellung der einfachen MySQL-Abfrage im Browser

Beispiel 2-3: laender_pdo.php – dritte Variante des Skripts in PDO-Syntax

```
<html>
  <head>
    <title>Einfache MySQL-Abfrage mit PHP, PDO-Version</title>
  </head>
  <body>
    <h1>Einige L&auml;nder, ihre Hauptst&auml;dte und Kontinente</h1>
    <table border="2" cellpadding="4">
      <tr>
        <th>Land</th>
        <th>Hauptstadt</th>
        <th>Kontinent</th>
      </tr>
<?php

  // Parameter für die Datenbankverbindung
  $host = "localhost";
  $user = "phpuser";
```

Beispiel 2-3: laender_pdo.php – dritte Variante des Skripts in PDO-Syntax (Fortsetzung)

```
  $pass = "PHP4ever";
  $db   = "geotest";
  // Verbindung herstellen und Datenbank auswählen
  $conn = new PDO ("mysql:host=$host;dbname=$db", $user, $pass);
  // MySQL-Abfrage
  $result = $conn->query ("SELECT * FROM laender ORDER BY land ASC");
  // Schleife zum Auslesen und Anzeigen der Ergebnisse
  while (list ($land, $hauptstadt, $kontinent) = $result->fetch()) {
    echo "<tr>\n";
    echo "<td>$land</td>\n";
    echo "<td>$hauptstadt</td>\n";
    echo "<td>$kontinent</td>\n";
    echo "</tr>\n";
  }
  // Datenbankverbindung schließen
  $conn = null;

?>
    </table>
  </body>
</html>
```

Distributionsware – LAMP unter openSUSE 10.2

Die Chancen, dass Ihre einigermaßen aktuelle Linux-Distribution bereits alle LAMP-Komponenten enthält oder zur Installation anbietet, stehen sehr gut. Dasselbe gilt für die wichtigsten BSD-Unix-Varianten wie zum Beispiel FreeBSD sowie für Mac OS X (dessen Unterbau Darwin ebenfalls ein BSD-Unix ist) – natürlich müsste es in diesen Fällen korrekt »BAMP« beziehungsweise »MAMP« statt LAMP heißen.

Jedes System und jede Distribution verwendet ihren eigenen Paketmanager mit jeweils anderen Konsolenbefehlen und/oder grafischen Hilfsprogrammen. Insofern ist es vollkommen ausgeschlossen, hier auf alle einzugehen. Als typisches Beispiel wird jedoch im Folgenden erläutert, wie Sie die LAMP-Komponenten in der recht weit verbreiteten Linux-Distribution openSUSE 10.2 aktivieren.

Starten Sie dazu als Erstes das zentrale Verwaltungstool YaST – entweder über das Menü der grafischen Oberfläche oder durch Eingabe von `yast` als `root` in einer Konsole. Beim grafischen Aufruf werden Sie aufgefordert, das `root`-Passwort einzugeben. Wählen Sie in YaST die Hauptkategorie *Software* und deren Unterpunkt *Software installieren oder löschen* aus. Daraufhin erscheint die Paketmanager-Oberfläche, die auch während der Neuinstallation von openSUSE zum Einsatz kommt.

→

Wählen Sie unter *Filter* den Eintrag *Schemata*, um die Paketlisten thematisch zu sortieren. Alle LAMP-Elemente befinden sich in der Paketgruppe *Web- und LAMP-Server* im Abschnitt *Serverfunktionen*. Wählen Sie die Gruppe zur Installation aus und aktivieren Sie gegebenenfalls in der Detailansicht noch einige der Pakete, die nicht automatisch hinzugefügt wurden. In Abbildung 2-2 sehen Sie die betreffende Bildschirmseite in der grafischen Variante.

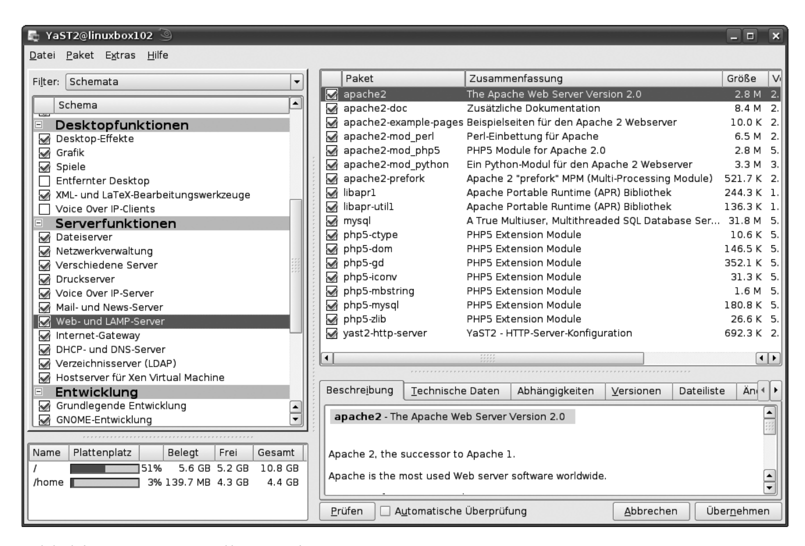


Abbildung 2-2: Installation der LAMP-Komponenten unter openSUSE 10.2

Wählen Sie zum Schluss die Option *Übernehmen*. Gegebenenfalls wird angezeigt, dass weitere Pakete benötigt werden, um Abhängigkeiten aufzulösen; diese Pakete werden automatisch ausgewählt. Nach der Installation steht Ihr LAMP-System sofort zur Verfügung, und Sie können die in den vorigen Abschnitten beschriebenen Apache-, MySQL- und PHP-Tests durchführen.

phpMyAdmin einrichten

Es gibt verschiedene Clientprogramme zum Durchsuchen, Editieren und Modifizieren von MySQL-Datenbanken. Mit dem MySQL-Server wird lediglich der Kommandozeilenclient mysql geliefert. Die MySQL AB bietet zusätzlich die beiden grafischen Tools *MySQL Administrator* und *MySQL Query Browser* an (siehe Anhang C).

Eine weitere beliebte Lösung ist das Open Source-Tool *phpMyAdmin*. Wie der Name schon sagt, ist es in PHP geschrieben und kann in jedem beliebigen Webbrowser ausgeführt werden, wobei es keine Rolle spielt, ob phpMyAdmin und der MySQL-Server auf demselben oder auf unterschiedlichen Rechnern ausgeführt wer-

den. Zudem wird phpMyAdmin von etlichen Webhostern bereitgestellt, um die in vielen Hosting-Tarifen enthaltenen MySQL-Datenbanken zu verwalten. Aus diesen Gründen habe ich mich dazu entschlossen, phpMyAdmin in diesem Buch als Standardwerkzeug zur Datenbankbearbeitung einzusetzen.

Sie finden die zurzeit aktuelle Version 2.10.1 auf der beiliegenden CD-ROM; eine neuere können Sie sich gegebenenfalls von der Website des Projekts (*http://www.phpmyadmin.net*) herunterladen.

Eine Installation im eigentlichen Sinn ist nicht erforderlich. Im Wesentlichen brauchen Sie die Archivdatei nur zu entpacken und unter einer URL in Ihrem PHP-fähigen Webserver verfügbar zu machen. Für Windows ist die ZIP-Version (*phpMyAdmin-2.10.1-all-languages.zip*) am besten geeignet; Sie können sie wie üblich mit WinZip oder XP-Bordmitteln entpacken. Für Linux eignet sich dagegen eher die *gzip*-Datei; verwenden Sie die übliche Anweisung zum Auspacken:

```
# tar -xzvf phpMyAdmin-2.10.1-all-languages.tar.gz
```

Anschließend sollten Sie das neue Verzeichnis *phpMyAdmin-2.10.0-all-languages* mithilfe einer `Alias`-Direktive in den Webspace Ihres Apache-Servers einbinden. Fügen Sie eine der beiden folgenden Zeilen, die natürlich noch an Ihr konkretes Verzeichnis angepasst werden muss, in Ihre *httpd.conf* ein:

```
# Linux
Alias /phpmyadmin /usr/local/phpMyAdmin-2.10.1-all-languages
# Windows
Alias /phpmyadmin C:/phpMyAdmin-2.10.1-all-languages
```

Die passende Stelle für diese Direktive ist der bereits vorhandene Block:

```
<IfModule alias_module>
   ...
</IfModule>
```

Es ist wichtig, die phpMyAdmin-URL vor unberechtigten Zugriffen zu schützen. Um die Benutzeranmeldung kann sich phpMyAdmin auf Wunsch selbst kümmern (siehe unten). Zusätzlich sollten Sie aber auch den URL-Pfad auf den einzelnen Rechner oder das lokale Netzwerk beschränken. Fügen Sie dazu die passenden Zeilen aus der nachstehenden Direktivenabfolge in Ihre Konfigurationsdatei ein:

```
<Location /phpmyadmin>
   Order Deny,Allow
   Deny from all
   # Nur lokaler Rechner
   Allow from 127.0.0.1
   # Alternative: Nur lokales Netz
   Allow from 192.168.0
   # Falls vorhanden, Adressbeschränkung UND Anmeldung beachten
   Satisfy All
</Location>
```

Starten Sie Apache neu, damit die Konfigurationsänderungen wirksam werden. Wenn Sie nun einen Browser öffnen und die phpMyAdmin-URL eingeben – zum Beispiel *http://localhost/phpmyadmin* –, wird die Fehlermeldung aus Abbildung 2-3 angezeigt. Klicken Sie den Link *Setup-Skript* an, um das empfohlene Konfigurationsskript aufzurufen. Es führt Sie mithilfe von Webformularen durch die Erzeugung der Konfigurationsdatei.



Abbildung 2-3: Meldung beim ersten Start von phpMyAdmin, die besagt, dass noch keine Konfigurationsdatei existiert

Normalerweise zeigt das Skript zwei Fehlermeldungen an. *Not secure Connection* – die Empfehlung, besser eine SSL-verschlüsselte Verbindung zu verwenden – können Sie auf dem eigenen Rechner ignorieren. Die andere ist dagegen wichtig: Das Skript benötigt ein Unterverzeichnis namens *config* im phpMyAdmin-Verzeichnis, für das der Webserver Schreibrechte besitzt. Auf Unix-Systemen funktioniert dies sehr einfach, über die Konsole beispielsweise mithilfe folgender Anweisungen:

```
# cd <phpMyAdmin-Verzeichnis>
# mkdir config
# chmod a+w config
```

Unter Windows können Sie zunächst versuchen, das Verzeichnis einfach zu erzeugen und die Seite im Browser neu zu laden. Wenn die Fehlermeldung weg ist, haben Sie es geschafft. Andernfalls müssen Sie auch hier die Berechtigung ändern. Rufen Sie dazu als Erstes den Task-Manager auf (Rechtsklick auf eine leere Stelle in der Taskleiste und *Task-Manager* wählen). Wechseln Sie auf die Registerkarte und ermitteln Sie, unter welchem Benutzernamen der Webserver *httpd.exe* läuft (meist *SYSTEM*). Wechseln Sie ins phpMyAdmin-Verzeichnis und geben Sie folgende Anweisung ein (`SYSTEM` gegebenenfalls durch den gefundenen Usernamen ersetzen):

```
> cacls config /e /t /g SYSTEM:F
```

Ein anschließender Reload müsste die Fehlermeldung auf beiden Systemplattformen verschwinden lassen, so dass Sie bereit sind, die Konfiguration zu ändern. Klicken Sie dazu unter *Servers* zunächst auf *Add*. Das Formular aus Abbildung 2-4 wird angezeigt.

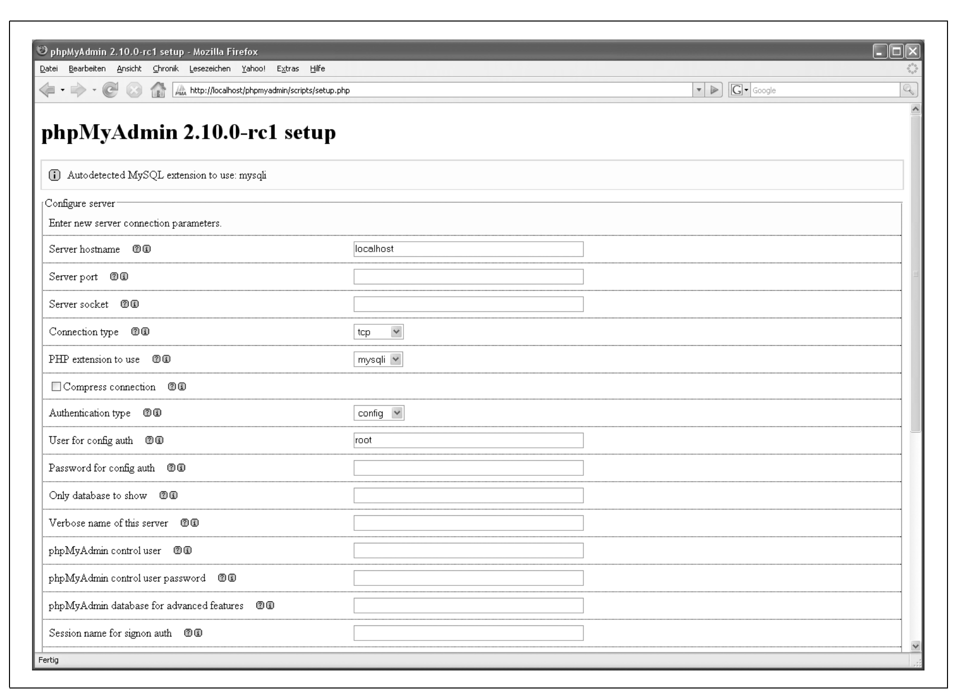


Abbildung 2-4: Das Konfigurationsformular von phpMyAdmin

Unter *Server hostname* kann in der Regel `localhost` stehen bleiben – es sei denn, Ihr MySQL-Server befindet sich auf einem anderen Rechner als phpMyAdmin. *Server port* oder *Server socket* brauchen Sie auch nur dann auszufüllen, wenn diese Werte vom Standard abweichen (siehe Kapitel 9). *Connection type* muss unter Windows sowie bei einem entfernten Rechner stets `tcp` sein; bei einem lokalen MySQL-Server auf einem Unix-Rechner können Sie auch `socket` wählen, um per Unix Domain Socket mit dem Server zu kommunizieren (verbessert die Performance und bei speziellen Einstellungen auch die Sicherheit ein wenig). Die Option *Compress connection* überträgt die Daten zwischen Web- und Datenbankserver komprimiert. Dies beschleunigt die Datenübertragung, verlangsamt aber die Verarbeitung und ist damit nur bei einem entfernten Datenbankserver und relativ langsamer Netzwerkanbindung zu empfehlen.

Der *Authentication type* bestimmt, wie die Benutzeranmeldung am MySQL-Server durchgeführt wird. Die verfügbaren Optionen sind:

config

User und Passwort werden fest in die Konfigurationsdatei eingetragen; verwenden Sie dazu die beiden Felder *User for config auth* beziehungsweise *Password for config auth*. Diese Option ist bequem, aber verhältnismäßig unsicher. Wenn Sie sie wählen, ist es besonders wichtig, den phpMyAdmin-Zugriff wie oben gezeigt auf Ihren Rechner oder auf das lokale Netzwerk zu beschränken. Zudem sollten Sie `root` auch dann nicht anmelden, wenn Sie Vollzugriff auf den MySQL-Server benötigen. Erzeugen Sie stattdessen gemäß der Anleitung in Kapitel 9 einen neuen MySQL-Benutzer mit den gewünschten Rechten und tragen Sie seinen Usernamen und sein Passwort ein.

http

Diese Option zeigt beim ersten Öffnen von phpMyAdmin innerhalb einer Sitzung den Authentifizierungsdialog des Browsers an. Hier können Sie jedes Mal einen anderen MySQL-User und sein Passwort eingeben. Diese Variante benötigt ein separates MySQL-Benutzerkonto, das Usernamen und Passwörter nachschlagen darf (siehe unten).

cookie

Funktioniert im Prinzip wie der Modus *http*. Intern werden die Anmeldedaten allerdings in einem Cookie (siehe Kapitel 8) gespeichert, was unter anderem ein echtes Abmelden ermöglicht.

signon

Dieser in Version 2.10 neu eingeführte Modus verlässt sich darauf, dass sich eine andere Webanwendung um die Anmeldung kümmert. Sie müssen einen Sessionnamen angeben, den sich phpMyAdmin mit dieser Anwendung teilt.

Wenn Sie den Typ *config* gewählt haben, müssen Sie unter *User for config auth* den Benutzernamen und unter *Password for config auth* das Passwort des MySQL-Benutzers angeben, der automatisch in phpMyAdmin ausgewählt werden soll. Für alle anderen Modi können Sie die beiden Felder dagegen frei lassen.

Für *http* oder *cookie* muss dagegen zunächst der Control User zum Auslesen der Anmeldedaten erstellt werden. Öffnen Sie dazu den Kommandozeilenclient mysql und geben Sie folgende Zeilen ein (den Usernamen *phpcontroller* und vor allem das Passwort *ctrlPHP* sollten Sie dabei ändern):

```
mysql> CREATE USER phpcontroller@localhost IDENTIFIED BY "ctrlPHP";
mysql> GRANT SELECT ON mysql.* TO phpcontroller@localhost;
mysql> FLUSH PRIVILEGES;
```

Nun können Sie den Usernamen und das Passwort unter *phpMyAdmin control user* beziehungsweise *phpMyAdmin control user password* eingeben.

Wenn Sie möchten, können Sie eine separate Datenbank anlegen, in der phpMyAdmin Verwaltungsinformationen speichern kann. Dies eröffnet einige zusäzliche Möglichkeiten. Erzeugen Sie diese Datenbank zum Beispiel im Kommandozeilenclient mittels

```
mysql> CREATE DATABASE phpmyadmin;
```

Geben Sie den Namen der Datenbank (hier `phpmyadmin`) unter *phpMyAdmin database for advanced features* ein.

Für den Anmeldetyp *signon* müssen Sie unter *Session name for signon auth* den Namen der Session angeben, den die andere beteiligte Anwendung vorgibt. Auch die *Login URL for signon auth* muss eingetragen werden.

Für alle Anmeldeverfahren außer *config* können Sie zuletzt eine *Logout URL* festlegen, zu der phpMyAdmin automatisch wechselt, wenn ein User den Logout-Link betätigt.

Überprüfen Sie zum Schluss noch einmal alle Ihre Eingaben und klicken Sie dann auf die Schaltfläche *Add* in der Zeile *Actions*. Wenn Sie alles richtig gemacht haben, erscheint wieder der Startbildschirm des Setup-Skripts mit der Informationszeile *New Server added*. Klicken Sie die Schaltfläche *Save* unter *Configuration* an, um die neue Konfigurationsdatei zu speichern. Daraufhin erscheint die Information *File saved*. Einige weitere Konfigurationsoptionen werden in Kapitel 4 kurz erwähnt.

Zum Abschluss müssen Sie die neu erzeugte Datei *config.inc.php* aus dem Unterverzeichnis *config* in das phpMyAdmin-Hauptverzeichnis verschieben. Das global beschreibbare Verzeichnis *config* sollten Sie danach aus Sicherheitsgründen löschen. Diese beiden Schritte lassen sich entweder in der grafischen Oberfläche Ihres Systems ausführen, oder aber Sie verwenden die Konsole. Wenn das Arbeitsverzeichnis Ihr phpMyAdmin-Ordner ist, funktioniert Letzteres unter Unix so:

```
# mv config/config.inc.php .
# rmdir config
```

Auf Windows-Rechnern lauten diese Zeilen wie folgt:

```
# move config\config.inc.php .
# rmdir config
```

Anschließend können Sie phpMyAdmin unter der URL *http://<Domainname>/phpmyadmin* im Browser aufrufen – auf dem lokalen Rechner zum Beispiel unter *http://localhost/phpmyadmin* beziehungsweise *http://127.0.0.1/phpmyadmin*. Falls Sie den Anmeldetyp *http* oder *cookie* gewählt haben, werden Sie nun nach Benutzername und Passwort gefragt.

Im linken Frame der phpMyAdmin-Oberfläche können Sie als Nächstes eine Datenbank auswählen – oder im rechten Hauptbereich einige allgemeine Informationen über den MySQL-Server abrufen. Wenn Sie die weiter oben erstellte Datenbank *geotest* auswählen und die Schaltfläche *Anzeigen* neben dem Tabellentitel *laender*

anklicken, wird der Inhalt der Tabelle wie in Abbildung 2-5 zu sehen angezeigt. Im Übrigen funktioniert phpMyAdmin überaus intuitiv, dennoch erfahren Sie in Kapitel 4 weitere Einzelheiten über seine Bedienung.

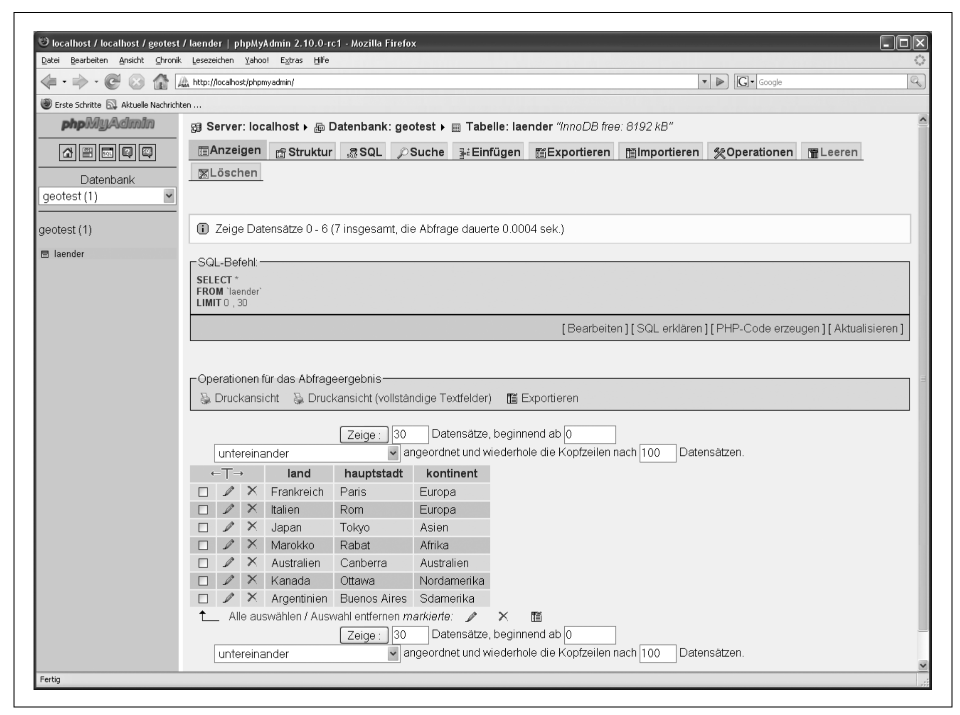


Abbildung 2-5: Der erste Test von phpMyAdmin

In diesem Kapitel:

- Die Datenbank erstellen
- Die PHP-Skripten
- Veröffentlichung und Test

KAPITEL 3

Die erste Webanwendung

Die eigentliche Kunst liegt viel weniger in der Kenntnis der Grundsätze als in der Art ihrer Anwendung.

Honoré de Balzac

Als erstes vollständiges Beispiel für eine PHP-Webanwendung wird in diesem Kapitel ein Online-Gewinnspiel entwickelt, das zur Eröffnung des Reisebüros veranstaltet wird. So lernen Sie anschaulich und übersichtlich die wichtigsten Grundlagen des Datenbankdesigns und der Programmierung von Webanwendungen kennen. In den nachfolgenden Kapiteln werden diese Themen systematischer erläutert und weiter vertieft.

Im Einzelnen enthält dieses Kapitel die folgenden drei Abschnitte:

Die Datenbank erstellen
: Entwurf und Einrichtung der zugrunde liegenden MySQL-Datenbank

Die PHP-Skripten
: Erläuterungen und vollständiger Code der PHP-Skripten für diese Anwendung

Veröffentlichung und Test
: Bereitstellung der Skripten durch den Webserver und Praxistest im Browser

Die Datenbank erstellen

Bei jeder datenbankbasierten Anwendung muss die Datenbank erstellt werden, bevor Sie mit dem Programmieren beginnen können. In diesem Abschnitt wird die Datenbank zunächst aus den Anforderungen der geplanten PHP-Anwendung entworfen, anschließend werden die praktischen Arbeitsschritte ihrer Erstellung gezeigt.

Der Datenbankentwurf

Der erste Schritt beim Entwurf einer Datenbank ist natürlich die Frage, welche Daten sie überhaupt enthalten soll. Zunächst sollten grundsätzliche Kategorien aufgestellt werden, anschließend können Sie überlegen, welche Details über sie gespeichert werden müssen. Erst danach kann die wichtige Entscheidung getroffen werden, in welche Tabellen Sie die Daten aufteilen.

Bei einer datenbankgestützten Webanwendung ist der Datenbedarf vor allem von der Funktionsweise der Anwendung abhängig. Deshalb folgt hier zunächst eine Übersicht über die geplante Anwendung:

- Auf der ersten Seite wird das Gewinnspiel als Webformular präsentiert. Es besteht aus vier Multiple-Choice-Fragen mit je drei Antwortmöglichkeiten. Die Fragen und Antworten sollen aus der Datenbank ausgelesen werden. Dabei werden die Antwortmöglichkeiten als Radio-Buttons (Optionsfelder) dargestellt, so dass für jede Frage immer nur eine Antwort ausgewählt werden kann. Darunter werden einige persönliche Daten erfragt: ein Benutzername, eine E-Mail-Adresse (für die eventuelle Gewinnbenachrichtigung) sowie die marketingrelevante Information, welche von vier angegebenen europäischen Großstädten der Benutzer in nächster Zeit besuchen möchte.
- Durch das Absenden des Formulars wird ein zweites PHP-Skript aufgerufen. Zunächst überprüft es, ob alle Formularfelder ausgefüllt wurden; falls nicht, verweist es automatisch auf das vorige Skript zurück. Ansonsten werden die Informationen in die Datenbank geschrieben; sollte dabei etwas schiefgehen, erfolgt die Weiterleitung zu einer Fehlermeldungsseite. Hat alles funktioniert, erhält der Benutzer eine kurze Dankeschön-Mitteilung.
- Das dritte Skript wird nicht von öffentlichen Benutzern der Website ausgeführt, sondern vom Betreiber der Site: Es zeigt nach dem Stichtag alle Einsender mit richtigen Lösungen an und wählt per Zufallsgenerator den Gewinner aus.

Offensichtlich müssen Daten aus folgenden Kategorien gespeichert werden:

- Fragen und Antworten für das Gewinnspiel
- persönliche Daten der Spielteilnehmer
- gewählte Antworten der Spielteilnehmer

Bei näherem Hinsehen bestehen die Fragen und Antworten aus folgenden Einzelaspekten:

- Fragetext
- drei Antworttexte
- Nummer der richtigen Antwort

Über die Spielteilnehmer selbst werden folgende Informationen gespeichert:

- Benutzername
- E-Mail-Adresse
- bevorzugte Stadt

Die Antworten der einzelnen Benutzer lassen sich mithilfe der folgenden Daten abspeichern:

- Benutzernummer
- Fragenummer
- Nummer der gewählten Antwort

Die Teilnehmerinformationen sowie die gegebenen Antworten bilden natürlicherweise je eine Tabelle. Die Fragen und Antworten selbst lassen sich dagegen nicht redundanzfrei in einer gemeinsamen Tabelle speichern, weil eine Frage je drei mögliche Antworten besitzt. Natürlich könnte man drei Spalten für die drei Antworten jeder Frage anlegen (zum Beispiel *antwort1*, *antwort2* und *antwort3*). Gemäß dem strengen Standard für relationale Datenbanken – den in Kapitel 5 besprochenen Normalisierungsregeln – gehören mehrere gleichartige Informationen aber nicht in einen Datensatz. Für ein sauberes Datenbankdesign gehören die Antworten also in eine separate Tabelle mit folgenden Spalten:

- Fragenummer
- Antwortnummer
- Antworttext

Diese Tabelle enthält demnach pro Frage drei Datensätze mit den verschiedenen Antwortmöglichkeiten. Die Tabelle mit den Fragen selbst wird dagegen folgende Daten enthalten:

- Fragenummer
- Fragetext
- Antwortnummer der korrekten Antwort

Abbildung 3-1 stellt die Abhängigkeiten zwischen den geplanten Tabellen übersichtlich dar.

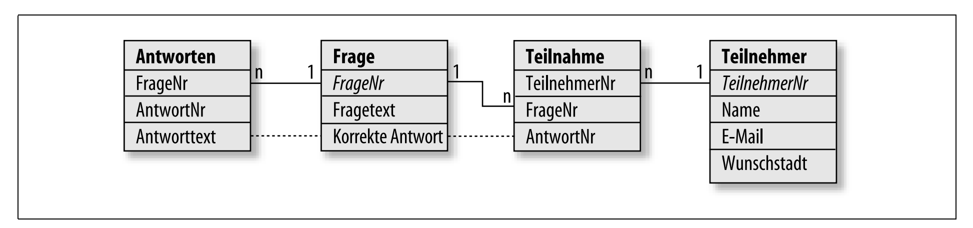


Abbildung 3-1: Beziehungen zwischen den Tabellen der Gewinnspielanwendung

Datenbank und Tabellen erstellen

Damit Sie Ihre Fähigkeiten beim Erstellen von SQL-Abfragen verbessern können, wird die Erstellung der Datenbank und ihrer Tabellen hier nicht für den komfortablen grafischen Client *phpMyAdmin*, sondern für das Konsolenprogramm *mysql* dokumentiert. Aber natürlich lassen sich die einzelnen Schritte auch mit einem grafischen Tool nachvollziehen.



Denken Sie auch hier wieder an die Zeichensatzanpassung – geben Sie nach dem Start des Kommandozeilenclients zunächst `SET NAMES utf8;` (Unix) beziehungsweise `SET NAMES cp850;` (Windows) ein.

Zuerst wird speziell für diese Anwendung ein eigener Datenbankbenutzer erstellt. Er heißt winuser, darf nur vom lokalen Rechner aus zugreifen und erhält das Passwort »G3w1nn3n«. Geben Sie dazu Folgendes ein:

```
mysql> CREATE USER winuser@localhost IDENTIFIED BY "G3w1nn3n";
mysql> GRANT ALL PRIVILEGES ON gewinnspiel.* TO winuser@localhost;
mysql> FLUSH PRIVILEGES;
```

Falls Sie noch MySQL 4.x verwenden, müssen Sie die `CREATE USER`-Zeile durch folgende Anweisung ersetzen:[1]

```
mysql> GRANT USAGE ON *.* TO winuser@localhost IDENTIFIED BY "G3w1nn3n";
```

Nachdem der berechtigte Benutzer existiert, muss die Datenbank erstellt werden:

```
mysql> CREATE DATABASE gewinnspiel;
```

Damit Sie bequem weiterarbeiten können (und nicht jedes Mal `gewinnspiel.`*`Tabellenname`* schreiben müssen), sollten Sie die neue Datenbank als Standard auswählen:

```
mysql> USE gewinnspiel
```

Nun können die einzelnen Tabellen erstellt werden. Dabei werden zwei empfehlenswerte Konventionen eingehalten, die Sie sich auch für eigene datenbankbasierte Anwendungen angewöhnen sollten:

1. Alle Tabellen erhalten das gleiche Namenspräfix (hier *gw* für »gewinnspiel«). Das ist beispielsweise nützlich, falls Sie später gezwungen sein sollten, die Tabellen zusammen mit anderen in einer gemeinsamen Datenbank zu speichern – in billigen Webhosting-Angeboten steht manchmal nur eine einzelne MySQL-Datenbank zur Verfügung.

---

1 In bestimmten Kombinationen älterer PHP- und MySQL-Versionen ist auch IDENTIFIED BY "Passwort" ein Problem, da sich die Passwortverschlüsselung in MySQL 4.1 geändert hat. Lesen Sie den Abschnitt »Benutzerverwaltung« in Kapitel 9 für weitere Hinweise.

2. Auch die Feldbezeichnungen jeder Tabelle erhalten je ein gemeinsames Präfix, um zu kennzeichnen, dass sie zur selben Tabelle gehören, und um Doppelbenennungen auszuschließen.[2]

Im ersten Arbeitsschritt wird die Tabelle *gw_fragen* mit den Fragetexten und der jeweiligen Nummer der richtigen Antwort erstellt:

```
mysql> CREATE TABLE gw_fragen (
    ->   fr_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   fr_frage VARCHAR(80),
    ->   fr_korrekt INT
    -> );
```

Die Fragenummer *fr_id* soll per `AUTO_INCREMENT` automatisch durchnummeriert werden. Die Angabe `PRIMARY KEY` macht dieses Feld zum Primärschlüssel, der den jeweiligen Datensatz eindeutig kennzeichnet. Der Primärschlüssel wird vor allem verwendet, wenn in anderen Tabellen Bezug auf Datensätze der aktuellen Tabelle genommen werden soll – genau dies macht das Wesen relationaler Datenbanken aus.

Die Tabelle *gw_antworten* soll für jede Frage drei Antwortmöglichkeiten enthalten. Dazu wird Folgendes eingegeben:

```
mysql> CREATE TABLE gw_antworten (
    ->   an_frage INT,
    ->   an_antwort INT,
    ->   an_text VARCHAR(50)
    -> );
```

Da sich keine andere Tabelle direkt auf die Inhalte dieser Tabelle beziehen wird, erhält sie keinen Primärschlüssel. Wenn Sie dennoch einen haben möchten, müssen Sie eine zusätzliche Spalte definieren, weil für keine der vorhandenen Spalten eindeutige Werte garantiert werden können. Das Feld *an_frage* verweist auf die Fragenummer, *an_antwort* ist die Nummer der jeweiligen Antwort (1–3), und *an_text* enthält den eigentlichen Antworttext. Jedes Paar aus diesen beiden Feldern ist in der Tabelle eindeutig; gemäß der Anleitung in Kapitel 5 könnten Sie daher auch die Kombination aus ihnen zum Primärschlüssel machen.

Als Nächstes werden die beiden Tabellen mit Werten gefüllt. Die vier Fragen, ihre möglichen Antworten und die jeweilige Nummer der richtigen Antwort finden Sie in Tabelle 3-1.

2 Manche objektrelationalen Mapper wie etwa Active Record aus dem Web-Framework Ruby on Rails besitzen etwas andere Benennungsvorgaben (siehe Anhang B).

Tabelle 3-1: Fragen und Antwortmöglichkeiten für das Gewinnspiel

Frage	Antwort 1	Antwort 2	Antwort 3	Korrekt
Wie heißt die Hauptstadt von Italien?	Ram	Rom	Bios	2
Welche dieser Hauptstädte hieß einst Lutetia?	Brüssel	London	Paris	3
Bratislava ist die Hauptstadt von ...?	Slowakei	Slowenien	Tschechien	1
Welche dieser Städte ist keine Hauptstadt?	Ljubljana	Istanbul	Oslo	2

Mithilfe der folgenden SQL-Abfrage können Sie die Werte in die Tabelle *gw_fragen* einfügen:

```
mysql> INSERT INTO gw_fragen (fr_frage, fr_korrekt) VALUES
    ->   ("Wie heißt die Hauptstadt von Italien?", 2),
    ->   ("Welche dieser Hauptstädte hieß einst Lutetia?", 3),
    ->   ("Bratislava ist die Hauptstadt von ...?", 1),
    ->   ("Welche dieser Städte ist keine Hauptstadt?", 2);
```

Da das Feld *fr_id* durch die Eigenschaft `AUTO_INCREMENT` automatisch ausgefüllt wird, können Sie die restlichen Felder vor `VALUES` explizit nennen und brauchen dann nur Werte für sie anzugeben.

Die Werte für *gw_antworten* werden durch folgende Abfrage eingefügt:

```
mysql> INSERT INTO gw_antworten VALUES
    ->   (1, 1, "Ram"),
    ->   (1, 2, "Rom"),
    ->   (1, 3, "Bios"),
    ->   (2, 1, "Brüssel"),
    ->   (2, 2, "London"),
    ->   (2, 3, "Paris"),
    ->   (3, 1, "Slowakei"),
    ->   (3, 2, "Slowenien"),
    ->   (3, 3, "Tschechien"),
    ->   (4, 1, "Ljubljana"),
    ->   (4, 2, "Istanbul"),
    ->   (4, 3, "Oslo");
```

Die Tabelle *gw_teilnehmer* mit den Teilnehmerstammdaten wird folgendermaßen erstellt:

```
mysql> CREATE TABLE gw_teilnehmer (
    ->   tn_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   tn_uname VARCHAR(40),
    ->   tn_email VARCHAR(50),
    ->   tn_interest VARCHAR(40)
    -> );
```

Als Letztes wird die Tabelle *gw_teilnahme* erzeugt, die die Antworten der einzelnen Spieler aufnehmen soll:

```
mysql> CREATE TABLE gw_teilnahme (
    ->   tl_tln INT,
    ->   tl_frag INT,
    ->   tl_antw INT
    -> );
```

Alle drei Felder sind numerisch und beziehen sich auf Felder anderer Tabellen: *tl_tln* verweist auf die Teilnehmer-ID (*tn_id*), den Primärschlüssel der Tabelle *gw_teilnehmer*. *tl_frag* bezieht sich auf die Nummer der beantworteten Frage, das heißt auf das Primärschlüsselfeld *fr_id* der Tabelle *gw_fragen*. *tl_antw* schließlich ist kein direkter Verweis, sondern speichert die vom Benutzer gegebene Antwort auf die jeweilige Frage; sie wird später bei der Auswertung mit dem Feld *fr_korrekt* aus der Tabelle *gw_fragen* verglichen.

Die PHP-Skripten

Nachdem die Datenbank eingerichtet ist, kann das Erstellen der PHP-Skripten beginnen. Ihre jeweiligen Aufgaben wurden weiter oben bereits erläutert. Bevor die Skripten selbst beschrieben werden, wird kurz erläutert, wie Daten aus Webformularen mithilfe von PHP gelesen werden.

HTTP-Grundwissen

Webserver wie Apache kommunizieren über das Anwendungsprotokoll *HTTP* (HyperText Transfer Protocol) mit Clientprogrammen – die meisten von ihnen sind Webbrowser wie Firefox oder Internet Explorer. Die genaue Definition der aktuellen HTTP-Version steht in RFC 2616 – online zum Beispiel unter *http://www.faqs.org/rfcs/rfc2616.html*.[3] Dieses Dokument ist allerdings sehr trocken und braucht normalerweise nur Entwickler zu interessieren, die den HTTP-Standard in die eigene Software implementieren möchten.

HTTP ist ein klartextbasiertes Internet-Anwendungsprotokoll; das heißt, Webserver und Browser verwenden (meist hinter den Kulissen) eine für Menschen lesbare Sprache für ihre Kommunikation. Das erleichtert Administratoren und Programmierern die Arbeit.

Wie alle Internet-Kommunikationsstandards gehört HTTP zur Anwendungsebene oberhalb der Vermittlungs- und Transportprotokolle TCP/IP. Um Webanwendungen mit PHP und MySQL zu schreiben, sollten Sie sich einigermaßen damit auskennen. Unter *http://www.galileocomputing.de/openbook/kit/itkomp13002.htm* finden Sie eine relativ ausführliche Einführung; es handelt sich um den entsprechenden Abschnitt aus der Onlinefassung meines Buchs *Kompendium der Informationstechnik*.

3 RFC heißt »Request for Comments« (Bitte um Kommentare). Seit über 30 Jahren werden die Standards des Internets und seiner Vorläufernetze in solchen öffentlich verfügbaren Dokumenten festgelegt. Dass diese Standards Allgemeingut sind und weder durch das Copyright einzelner Firmen noch durch die unsäglichen Softwarepatente beschränkt werden, ist einer der wichtigsten Gründe für den langfristigen Erfolg der Internetprotokolle.

Üblicherweise sendet der Client eine Anfrage an den Server – und zwar sobald Sie eine URL in die Adresszeile Ihres Browsers eintippen oder einen Link anklicken. Eine HTTP-Anfrage sieht beispielsweise folgendermaßen aus:

```
GET /seiten/info.html HTTP/1.1
Accept: */*
Accept-Language: de, en-US
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Windows; U; WinNT4.0; de-DE; rv:1.7.5) Gecko/
20041108 Firefox/1.0
Host: www.test.local
Connection: Keep-Alive
```

Die erste Zeile ist der eigentliche Befehl. Er besteht aus der HTTP-Methode – hier `GET` –, dem Pfad der angeforderten Ressource (*info.html* im Website-Unterverzeichnis */seiten*) sowie der HTTP-Protokollversion (standardmäßig HTTP/1.1). Die Methode `GET` fordert eine Ressource vom Server an. Eine andere bekannte HTTP-Methode ist zum Beispiel `POST`; sie kann zusätzlich größere Datenmengen – meist aus Webformularen – an den Webserver senden.

Alle anderen Zeilen sind *HTTP-Header*, die dem Server weitere Aspekte über den Client und die Anfrage mitteilen:

- `Accept` gibt eine Liste von MIME-Typen der Dokumentarten an, die der Client akzeptiert. MIME-Typen haben das Format `Haupttyp/Untertyp` – etwa `text/html` für HTML-Dokumente oder `image/jpeg` für JPEG-Bilder. `*/*` bedeutet, dass der Client alle Arten von Dokumenten annimmt.
- `Accept-Language` listet die ISO-Kürzel der bevorzugten Sprachen des Clients auf. Webserver wie Apache beherrschen eine Technik namens *Content-Negotiation*, die die `Accept*`-Header auswertet und Dokumente in der bevorzugten Sprache, dem bevorzugten Dateityp oder dem gewünschten Zeichensatz an Clients ausliefern kann. In unserem Beispiel werden Deutsch und US-Englisch gewünscht.
- `Accept-Encoding` gibt an, dass der Browser komprimierte Ressourcen verarbeiten kann, und listet die einzelnen Komprimierungsformate auf. Der Browser im Beispiel versteht GNU-Zip (`gzip`) und ZIP (`deflate`).
- `User-Agent` ist die Selbstidentifikation des Clients – hier handelt es sich um den Browser Firefox unter Windows NT.
- `Host` ist der wichtigste Anfrage-Header; in HTTP/1.1 ist er vorgeschrieben. Auf einem Serverrechner können mehrere Websites mit eigenen Domainnamen liegen, die als *virtuelle Hosts* bezeichnet werden. Damit der Server weiß, welche Site angefordert wird, benötigt er diesen Header.
- `Connection` gibt an, ob die Verbindung zwischen Client und Server bestehen bleiben soll (`keep-alive`) oder geschlossen wird (`close`). Das Offenhalten der Verbindung ermöglicht das schnellere Nachladen verknüpfter Dateien – wie beispielsweise Bilder, die in eine Webseite eingebettet wurden.

Auf eine solche Anfrage antwortet der Webserver mit einer HTTP-Antwort (Response). Diese besteht aus einer Statuszeile, mehreren Antwort-Headern, einer Leerzeile und schließlich dem Body, der die eigentliche, vom Client angeforderte Ressource liefert. Hier ein Beispiel für die HTTP-Server-Antwort auf eine GET-Anfrage:

```
HTTP/1.1 200 OK
Date: Mon, 26 Feb 2007 12:10:42 GMT
Server: Apache/2.0.53 (Unix)
Last-Modified: Thu, 22 Feb 2007 13:28:38 GMT
ETag: "39a4cb-5a3-6ef34d10"
Accept-Ranges: bytes
Content-Length: 2092
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//Transitional//EN">
<html>
<head>
[...]
```

Die Statuszeile gibt Auskunft über die HTTP-Version (hier 1.1) sowie über Erfolg oder Misserfolg der Anfrage. Der Status `200 OK` bedeutet, dass die Anfrage mit der Lieferung der gewünschten Ressource beantwortet werden kann. Andere wichtige Statusmeldungen sind etwa `404 Not Found` (Ressource nicht gefunden; meist bei fehlerhaften URL-Eingaben oder veralteten Links), `500 Internal Server Error` (könnte Ihnen bei Fehlern in der PHP-Programmierung begegnen) oder die diversen `3xx`-Weiterleitungen.

Auch bei der Antwort sind die restlichen Zeilen Header-Felder; die hier verwendeten bedeuten Folgendes:

- `Date` gibt Datum und Uhrzeit der Lieferung an.
- `Server` ist die Selbstidentifikation der Webserver-Software. Bei Apache wird ihre Ausführlichkeit durch die Direktive `ServerTokens` gesteuert. Die Extreme reichen von `ProductOnly` (also `Apache`) bis `Full` (zum Beispiel `Apache/2.2.4 (Unix) PHP/5.2.2`).
- `Last-Modified` enthält Datum und Uhrzeit der letzten Änderung. Browser- und Proxy-Caches nutzen diese Angabe, um zu entscheiden, ob die zwischengespeicherte Version noch gültig ist. Für solche Prüfungen existiert übrigens die spezielle HTTP-Anfragemethode `HEAD`, die nur die Header, aber nicht die eigentliche Ressource anfordert.
- `ETag` (Entity-Tag) ist ein aus verschiedenen Angaben berechneter Hash-Wert, der die Identität eines Dokuments über die URL oder den Dateinamen hinaus überprüft. Auch dies ist für Caching-Zwecke wichtig.

- `Accept-Ranges: bytes` gibt an, dass der Server Anfragen nach Teilbereichen einer Ressource akzeptiert. Das macht sich beispielsweise der Adobe (Acrobat) Reader zunutze, um umfangreiche PDF-Dokumente stückweise anzufordern.
- `Content-length` ist die Größe des Bodys (also der gelieferten Datei) in Byte.
- `Connection` entspricht dem gleichnamigen Anfrage-Header. Der Server gibt an, ob er das Offenhalten der Verbindung bestätigt (`Keep-Alive`) oder ablehnt (`close`).
- `Content-Type` ist der wichtigste Antwort-Header. Er gibt den Datentyp (MIME-Typ) der gelieferten Ressource an, damit der Client weiß, wie er die empfangenen Daten behandeln soll.

Nach einer Leerzeile folgt der Body, also der eigentliche Inhalt der Antwort. In unserem Beispiel wurde er auf die ersten drei Zeilen gekürzt.

Formulardaten lesen

Eine der wichtigsten Fähigkeiten von Webserver-Programmiersprachen ist das Auslesen und Verarbeiten von Daten aus HTML-Formularen. Sie bilden die Benutzeroberfläche von Webanwendungen. Deshalb soll an dieser Stelle kurz erläutert werden, wie Formulare aufgebaut sind und wie sich die entsprechenden Benutzereingaben mit PHP auslesen lassen.

Grundsätzlich wird jedes HTML-Formular von folgenden Tags umschlossen:

```
<form action="URL" method="HTTP-Methode">
  ...
</form>
```

Die URL bestimmt, wohin das Formular versandt wird, wenn ein Benutzer den Absendeknopf betätigt. In aller Regel handelt es sich um die Adresse eines serverseitigen Skripts (zum Beispiel PHP), das mit den Formulardaten umgehen kann.

Die HTTP-Methode kann `"get"` oder `"post"` sein. Ihre Bedeutung wurde im vorigen Abschnitt bereits angesprochen: Bei `"get"` werden die Formulardaten durch ein Fragezeichen getrennt an die URL der HTTP-Anfrage angehängt. Eine solche URL lässt sich beispielsweise verlinken – Sie könnten folgendermaßen einen Link auf die Google-Suche nach `"mysql"` setzen:

```
<a href="http://www.google.de/search?q=mysql">MySQL suchen</a>
```

`"post"` versendet die Daten dagegen separat. Dadurch ist diese Methode für den Versand beliebig großer Datenmengen geeignet.

Zwischen `<form>` und `</form>` können Sie beliebiges HTML einfügen, vor allem Formularelemente. Die meisten von ihnen werden durch das `<input>`-Tag bereitgestellt. Die einfachste Form ist ein Texteingabefeld mit folgender Grundsyntax:

```
<input type="text" name="feldname" />
```

Was ein Benutzer in dieses Feld einträgt, wird beim Versenden als `feldname=inhalt` zu den Formulardaten hinzugefügt. Das folgende Beispiel definiert ein Feld namens `user` mit einer Anzeigebreite von 40 Zeichen; die maximale Eingabelänge beträgt 50 Zeichen:

```
<input type="text" name="user" size="40" maxlength="50" />
```

Ein anderes Beispiel sind Radio-Buttons, die die Auswahl einer einzelnen Option aus mehreren ermöglichen. Dazu besitzen sie denselben Wert für das Attribut `name` und unterschiedliche Werte für `value` – den Wert, der übertragen wird, falls jemand die entsprechende Option auswählt. Daraus ergibt sich folgende Syntax:

```
<input type="radio" name="gruppenname" value="wert1" />Beschriftung 1
<input type="radio" name="gruppenname" value="wert2" />Beschriftung 2
...
```

Hier ein Beispiel zur Auswahl einer Zahlungsmethode:

```
Wie m&ouml;chten Sie zahlen?
<input type="radio" name="zahlung" value="kred" />Kreditkarte<br />
<input type="radio" name="zahlung" value="bank" />Bankeinzug<br />
<input type="radio" name="zahlung" value="rech" />Rechnung<br />
<input type="radio" name="zahlung" value="nach" />Nachnahme
```

Zum Abschicken des Formulars an die Action-URL wird ebenfalls ein `<input>`-Element verwendet, diesmal mit dem Attribut `type="submit"`. `value` enthält in diesem Fall die Beschriftung, zum Beispiel:

```
<input type="submit" value="Abschicken" />
```

Betrachten Sie das folgende kleine Formular als Beispiel für die Erläuterung, wie Formulardaten in PHP eingelesen werden:

```
<form action="umfrage.php" method="get">
  Wohin m&ouml;chten Sie reisen?<br />
  <input type="radio" name="ziel" value="Paris" />Paris<br />
  <input type="radio" name="ziel" value="K&ouml;ln" />K&ouml;ln<br />
  <input type="radio" name="ziel" value="London" />London<br />
  <br />
  Ihre E-Mail-Adresse:
  <input type="text" name="mail" size="40" maxlength="50" />
  <br />
  <input type="submit" value="Abschicken" />
</form>
```

Angenommen, ein Benutzer wählt *Paris*, gibt die E-Mail-Adresse *klaus@example.com* ein und klickt auf *Abschicken*. Da die Formularversandmethode `GET` ist, sendet der Browser eine Anfrage-URL wie diese:

```
http://www.test.local/umfrage.php?ziel=Paris&mail=klaus@example.com
```

In dem Skript *umfrage.php* sollen diese Daten ausgelesen und verarbeitet werden. Grundsätzlich stehen GET-Felder in PHP als Elemente der speziellen Array-Variab-

len $_GET zur Verfügung, während POST-Felder automatisch in $_POST gespeichert werden. Sie könnten die beiden Felder also zum Beispiel folgendermaßen auslesen und ihren Inhalt ausgeben:

```
<?php

  $ziel = $_GET['ziel'];
  $mail = $_GET['mail'];

  echo ("Zur Kontrolle: Sie m&ouml;chten nach $ziel fahren.<br />");
  echo ("Das Gesamtergebnis der Umfrage werden wir an Ihre E-Mail-Adresse $mail
  senden.");

?>
```

Die oben gezeigte Eingabe hat die folgende Ausgabe zur Folge:

> Zur Kontrolle: Sie möchten nach Paris fahren.
>
> Das Gesamtergebnis der Umfrage werden wir an Ihre E-Mail-Adresse klaus@example.com senden.

Normalerweise muss für jedes Formularfeld überprüft werden, ob der Benutzer überhaupt etwas eingegeben oder ausgewählt hat. Das ist relativ lästig, vor allem weil sich »leere« Elemente von $_GET und $_POST je nach PHP-Version unterschiedlich verhalten. Aus diesem Grund bietet sich die Definition der folgenden Funktion an, die das Problem grundsätzlich löst:

```
function cgi_param ($feld, $default="") {
  // Variable zunächst auf Default-Wert setzen
  $var = $default;
  if (isset($_GET[$feld]) && $_GET[$feld] != "") {
    // GET-Feld gefunden
    $var = $_GET[$feld];
  } elseif (isset($_POST[$feld]) && $_POST[$feld] != "") {
    // POST-Feld gefunden
    $var = $_POST[$feld];
  }
  // Ermittelten Wert zurückgeben
  return $var;
}
```

Der Name cgi_param() wurde gewählt, weil die Funktion von der Perl-CGI-Funktion param() inspiriert wurde. Aufgerufen wird sie mit dem Namen des Felds, dessen Inhalt Sie auslesen möchten, und mit einem Vorgabewert für den Fall, dass das Feld nicht ausgefüllt wurde oder nicht existiert. Das folgende Beispiel speichert in der Variablen $ziel den Inhalt des Felds ziel oder notfalls einen leeren String:

```
$ziel = cgi_param ("ziel", "");
```

Da $default in der Funktionsdefinition ohnehin den automatischen Vorgabewert "" erhält, können Sie diesen Aufruf auch noch kürzer schreiben:

```
$ziel = cgi_param ("ziel");
```

Innerhalb der Funktion wird die spätere Ergebnisvariable $var zunächst auf den Standardwert $default gesetzt. Anschließend wird überprüft, ob der gewünschte Datenname als GET- oder aber als POST-Feld existiert (isset()) und einen anderen Wert als "" besitzt. Nur in diesem Fall wird der Inhalt des Felds in $var gespeichert. Zum Schluss wird auf jeden Fall $var zurückgegeben, so dass Sie entweder den gewünschten Feldwert oder Ihren erwarteten Vorgabewert erhalten.

Diese Funktion wird in vielen Skripten in diesem Buch verwendet. In Kapitel 8 erfahren Sie auch, wie Sie solche Hilfsmittel in externe Dateien auslagern und bei Bedarf importieren können.

Das Gewinnspielformular

Die erste Datei der Gewinnspielanwendung, *spiel.php*, stellt das Webformular mit den Quizfragen zur Verfügung. Dazu werden die Fragen und die zugehörigen Antworttexte aus der Datenbank gelesen und als Formular angezeigt.

Zunächst einmal wird überprüft, ob das GET-Feld *fehler* gesetzt ist. Dies kommt nicht durch eine Benutzereingabe zustande, sondern wird automatisch durch das nächste Skript hervorgerufen, falls dieses eine vergessene Eingabe im Formular bemerkt. Falls *fehler* nicht den Vorgabewert 0 hat (konkret wird bei einem Fehler der Wert 1 gesetzt), zeigt das Skript eine zusätzliche Meldung an:

```
<?php

   // Wurde die Seite nach einem Eingabefehler erneut aufgerufen?
   $fehler = cgi_param ("fehler", 0);
   if ($fehler) {

?>
<p><font color="#FF0000">Bitte alles vollst&auml;ndig ausf&uuml;llen!</font></p>
<?php

   }

?>
```

Wie Sie sehen, lässt sich das Schließen und Wiederöffnen von <?php ... ?>-Blöcken innerhalb der if()-Verschachtelung verwenden. Dadurch kann man oft auf lästige echo-Anweisungen verzichten. Sie können optionalen HTML-Code einfach als solchen hinschreiben.

Nachdem wie üblich die Verbindung zum Datenbankserver hergestellt und die Datenbank ausgewählt wurde, erfolgt die Ausgabe der Daten in das Formular. Die erste Datenbankabfrage wählt die benötigten Bestandteile der Frage aus: Fragenummer (*fr_id*) und Fragetext (*fr_frage*). Der SQL-Code sieht folgendermaßen aus:

```
SELECT fr_id, fr_frage FROM gw_fragen ORDER BY fr_id ASC
```

Wenn Sie diese Abfrage im Kommandozeilenclient ausführen, erhalten Sie folgende Ausgabe:

```
+-------+---------------------------------------------+
| fr_id | fr_frage                                    |
+-------+---------------------------------------------+
|     1 | Wie heißt die Hauptstadt von Italien?       |
|     2 | Welche dieser Hauptstädte hieß einst Lutetia? |
|     3 | Bratislava ist die Hauptstadt von ...?      |
|     4 | Welche dieser Städte ist keine Hauptstadt?  |
+-------+---------------------------------------------+
```

Die Frage wird als normaler Text ausgegeben, anschließend werden mithilfe der folgenden Abfrage die zugehörigen Antworten ausgelesen:

```
SELECT an_antwort, an_text FROM gw_antworten WHERE an_frage=$fr_id
ORDER BY an_antwort ASC
```

Es werden also Antwortnummer (*an_antwort*) und Antworttext (*an_text*) aus denjenigen Datensätzen ausgewählt, in denen *an_frage* (zugeordnete Frage) der aktuellen Fragenummer `$fr_id` entspricht. Anschließend werden die Antwortmöglichkeiten als Radio-Buttons dargestellt. Das Attribut `name` bildet die Fragenummer mit vorangestelltem `"f"`, als `value` wird die jeweilige Antwortnummer eingetragen:

```
echo "<input type=\"radio\" name=\"f$fr_id\" value=\"$an_antwort\" />
$an_text<br />";
```



In SQL-Abfragen werden spezielle Schlüsselwörter, die das Standardverhalten einer Anweisung modifizieren, als *Klauseln* (englisch *clauses*) bezeichnet. Die wichtigste und bekannteste von allen ist die `WHERE`-Klausel; sie beschränkt die Wirkung der Abfrage auf diejenigen Datensätze, die dem angegebenen Kriterium entsprechen.

In Beispiel 3-1 sehen Sie das vollständige Listing in klassischer mysql-Syntax.

Beispiel 3-1: spiel.php – das Formular zur Teilnahme am Gewinnspiel

```
<?php

  function cgi_param ($feld, $default="") {
    // Variable zunächst auf Default-Wert setzen
    $var = $default;
    if (isset($_GET[$feld]) && $_GET[$feld] != "") {
      // GET-Feld gefunden
      $var = $_GET[$feld];
    } elseif (isset($_POST[$feld]) && $_POST[$feld] != "") {
      // POST-Feld gefunden
      $var = $_POST[$feld];
    }
    // Ermittelten Wert zurückgeben
    return $var;
  }

?>
```

Beispiel 3-1: spiel.php – das Formular zur Teilnahme am Gewinnspiel (Fortsetzung)

```
<html>
<head>
<title>Gewinnspiel</title>
<meta http-equiv="Content-type" content="text/html;charset=iso-8859-1" />
</head>
<body>
<h1>Gewinnspiel</h1>
<p>Beantworten Sie die folgenden Fragen und gewinnen Sie eine All-inclusive-
Wochenendreise in eine europ&auml;ische Gro&szlig;stadt aus unserem Angebot!</p>
<?php

   // Wurde die Seite nach einem Eingabefehler erneut aufgerufen?
   $fehler = cgi_param ("fehler", 0);
   if ($fehler) {

?>
<p><font color="#FF0000">
Bitte alles vollst&auml;ndig ausf&uuml;llen!</font></p>
<?php

   }

?>
<form action="teilnahme.php" method="post">
<?php

  // Verbindungsparameter
  $host = "localhost";
  $user = "winuser";
  $pass = "G3w1nn3n";
  $db   = "gewinnspiel";

  // Verbindung zum MySQL-Server herstellen
  $conn = mysql_connect ($host, $user, $pass);

  // Datenbank auswählen
  mysql_select_db ($db);

  // Abfrage senden
  $fr_query = mysql_query
      ("SELECT fr_id, fr_frage FROM gw_fragen ORDER BY fr_id ASC");

  // Zeilen lesen und Fragen stellen
  while (list ($fr_id, $fr_frage) = mysql_fetch_row ($fr_query)) {
    // Fragetext ausgeben
    echo "<b>$fr_id. $fr_frage</b><br /><br />";
    // Antworten holen
    $an_query = mysql_query ("SELECT an_antwort, an_text FROM
        gw_antworten WHERE an_frage=$fr_id ORDER BY an_antwort ASC");
    // Radio-Buttons und Antworten ausgeben
```

Beispiel 3-1: spiel.php – das Formular zur Teilnahme am Gewinnspiel (Fortsetzung)

```
    while (list ($an_antwort, $an_text) = mysql_fetch_row ($an_query)) {
      echo "<input type=\"radio\" name=\"f$fr_id\"
                  value=\"$an_antwort\" /> $an_text<br />";
    }
    echo "<br />";
  }

  // Datenbankverbindung schließen
  mysql_close();

?>
<h2>Pers&ouml;nliche Angaben</h2>
<table border="0" cellpadding="4">
<tr>
<td>Benutzername:</td>
<td colspan="3"><input type="text" name="uname" />
</tr>
<tr>
<td>E-Mail:</td>
<td colspan="3"><input type="text" name="email" />
</tr>
<tr>
<td colspan="4">Welche dieser St&auml;dte w&uuml;rden Sie bald am
liebsten besuchen?</td>
</tr>
<tr>
<td><input type="radio" name="wish" value="1" />Paris</td>
<td><input type="radio" name="wish" value="2" />London</td>
<td><input type="radio" name="wish" value="3" />Istanbul</td>
<td><input type="radio" name="wish" value="4" />Rom</td>
</tr>
</table>
<input type="submit" value="Abschicken" />
</form>
</body>
</html>
```

Um die objektorientierte *mysqli*-Syntax anzuwenden, müssen Sie folgende Stellen des Listings ändern (auf der beiliegenden CD-ROM finden Sie auch diese Fassung sowie die PDO-Variante):

- Verbindungsaufnahme und Datenbankauswahl:

```
// Verbindungsparameter
$host = "localhost";
$user = "winuser";
$pass = "G3w1nn3n";
$db   = "gewinnspiel";

// Verbindung herstellen und Datenbank auswählen
$conn = new mysqli ($host, $user, $pass, $db);
```

- Fragen aus der Datenbank lesen:

  ```
  $fr_query = $conn->query
       ("SELECT fr_id, fr_frage FROM gw_fragen ORDER BY fr_id ASC");

  while (list ($fr_id, $fr_frage) = $fr_query->fetch_row()) {
    ...
  }
  ```

- Antworten aus der Datenbank lesen:

  ```
  $an_query = $conn->query ("SELECT an_antwort, an_text FROM
       gw_antworten WHERE an_frage=$fr_id ORDER BY an_antwort ASC");

  while (list ($an_antwort, $an_text) = $an_query->fetch_row()) {
    ...
  }
  ```

- Datenbankverbindung schließen:

  ```
  $conn->close();
  ```

Für PHP Data Objects muss der gesamte datenbankrelevante Block modifiziert werden. Das Hauptproblem besteht darin, dass für verschachtelte Abfragen innerhalb einer Schleife – hier also für alle Antworten zur jeweiligen Frage – nicht dieselbe Datenbankverbindung verwendet werden darf. Innerhalb der Fragenschleife wird daher die zusätzliche Verbindung `$conn2` zum Auslesen der Antworten geöffnet. Ein Vorteil von PDO ist dagegen, dass die Methode `query()` selbst als Schleifenbedingung einer `foreach()`-Schleife dienen kann; ein zusätzliches `fetch()` ist nicht nötig. Zudem werden PDO-Blöcke üblicherweise komplett in einen `try{}`-Block gesetzt, um mittels `catch()` eventuelle Fehler vom Typ `PDOException` abzufangen. Ersetzen Sie den gesamten PHP-Block für die PDO-Lösung durch folgenden Inhalt (das komplette Skript finden Sie wieder auf der CD-ROM):

```
<?php

  // Verbindungsparameter
  $host = "localhost";
  $user = "winuser";
  $pass = "G3w1nn3n";
  $db   = "gewinnspiel";

  try {
    // Verbindung herstellen und Datenbank auswählen
    $conn = new PDO ("mysql:host=$host; dbname=$db", $user, $pass);

    // Abfrage für Fragen formulieren
    $fr_query =
         "SELECT fr_id, fr_frage FROM gw_fragen ORDER BY fr_id ASC";

    foreach ($conn->query($fr_query) as $fr_row) {
      // Frageteile auslesen
      $fr_id = $fr_row['fr_id'];
      $fr_frage = $fr_row['fr_frage'];
```

```
        // Frage ausgeben
        echo "<b>$fr_id. $fr_frage</b><br /><br />";

        // Neues Verbindungsobjekt für verschachtelte Abfrage!
        $conn2 = new PDO
                 ("mysql:host=$host; dbname=$db", $user, $pass);
        // Abfrage für Antworten zur aktuellen Frage formulieren
        $an_query = "SELECT an_antwort, an_text FROM gw_antworten
                     WHERE an_frage=$fr_id ORDER BY an_antwort ASC";

        foreach ($conn2->query($an_query) as $an_row) {
          // Antwortteile auslesen
          $an_antwort = $an_row['an_antwort'];
          $an_text = $an_row['an_text'];
          // Radio-Button und Antwort ausgeben
          echo "<input type=\"radio\" name=\"f$fr_id\"
                      value=\"$an_antwort\" /> $an_text<br />";
        }
        // Antworten-Verbindung schließen
        $conn2 = null;
        echo "<br />";
      }
      // Fragen-Verbindung schließen
      $conn = null;
    } catch (PDOException $e) {
      print "ERR! ".$e->getMessage();
    }

  ?>
```

Daten in die Datenbank übernehmen

Das zweite Skript trägt den Namen *teilnahme.php*; es hat die Aufgabe, die Antworten und Eingaben der Benutzer in die Datenbanktabellen *gw_teilnehmer* und *gw_teilnahme* einzufügen. Zunächst werden mithilfe der bereits besprochenen Funktion `cgi_param()` die verschiedenen Formulareingaben ausgelesen. Zuvor wird die Hilfsvariable `$korrekt` auf 1 gesetzt; das Skript geht zunächst davon aus, dass die Eingaben vollständig sind. Die Antworten auf die vier Quizfragen werden mithilfe der folgenden Schleife ermittelt:

```
for ($i = 1; $i <= 4; $i++) {
  $antwort[$i] = cgi_param ("f$i", 0);
  if ($antwort[$i] == 0) {
    $korrekt = 0;
  }
}
```

Die jeweilige Feldbezeichnung wird aus dem Buchstaben `"f"` und dem aktuellen Wert des Schleifenzählers `$i` gebildet. Sollte eine Antwort fehlen (Vorgabewert 0), wird `$korrekt` auf 0 gesetzt – das Formular wurde nicht vollständig ausgefüllt.

Danach werden auch die restlichen Eingaben – Benutzername, E-Mail-Adresse und gewünschte Stadt – gelesen:

```
$uname = cgi_param ("uname", "");
$email = cgi_param ("email", "");
$interest = cgi_param ("wish", 0);
if ($uname == "" || $email == "" || $interest == 0) {
  $korrekt = 0;
}
```

Sollte `$korrekt` nach der Auswertung aller Eingaben den Wert 0 haben, wird automatisch zum Frage-Skript *spiel.php* zurückgesprungen. Dazu wird mithilfe der PHP-Funktion `header()` der HTTP-Header *Location* gesetzt:

```
if (!$korrekt) {
  header ("Location: spiel.php?fehler=1");
}
```



Beachten Sie in diesem Zusammenhang, dass die Funktion `header()` nur benutzt werden kann, wenn noch *kein einziges HTML-Zeichen* erzeugt wurde. Sie muss also in einem PHP-Block ganz am Anfang der Datei stehen; vor dessen Beginnsequenz `<?php` darf nicht einmal ein Leerzeichen stehen.

Der *Location*-Header sorgt für eine automatische Weiterleitung, indem er den Browser anweist, die angegebene URL zu laden. Das GET-Feld `fehler=1` wird hier einfach programmgesteuert an die URL angehängt. Auf der Formularseite wird dadurch, wie oben beschrieben, die zusätzliche Fehlermeldung angezeigt. Der Rest des Programmcodes befindet sich im `else`-Teil der obigen Fallentscheidung. Die Ausführung von PHP-Skripten wird nämlich nach einer HTTP-Weiterleitung keineswegs abgebrochen. Deshalb muss auf diese Weise sichergestellt werden, dass die unvollständigen Daten nicht versehentlich in der Datenbank gespeichert werden.

Als Erstes werden die drei Benutzerinformationen in die Tabelle *gw_teilnehmer* geschrieben:

```
mysql_query ("INSERT INTO gw_teilnehmer (tn_uname, tn_email,
     tn_interest) VALUES (\"$uname\", \"$email\", $interest)");
```

Bei mysqli und PDO heißt es wie üblich `$conn->query()` statt `mysql_query()`. Als Nächstes wird mithilfe von `mysql_affected_rows()` sichergestellt, dass die Eintragung in die Datenbank tatsächlich stattgefunden hat – die Funktion zählt die geänderten Zeilen von Änderungsabfragen. Andernfalls wird die Fehlerseite *error.html* geladen:

```
if (mysql_affected_rows() == 0) {
  header ("Location: error.html");
}
```

Beachten Sie, dass affected_rows in mysqli-Syntax keine Methode, sondern eine Eigenschaft des Verbindungsobjekts ist – Sie dürfen keine Parameterklammern dahinter setzen. Konkret lauten die Zeilen in der mysqli-Fassung folgendermaßen:

```
if ($conn->affected_rows == 0) {
  header ("Location: error.html");
}
```

Aus den oben erwähnten Gründen wird der Rest des Codes wiederum in einen (verschachtelten) else-Block gesetzt. Zunächst wird die ID des soeben eingefügten Teilnehmers ermittelt, um dessen Antworten registrieren zu können:

```
$query = mysql_query
     ("SELECT tn_id from gw_teilnehmer WHERE tn_email=\"$email\"");
list ($id) = mysql_fetch_row ($query);
```

Die vier Antworten werden nun in einer Schleife in die Datenbank geschrieben. Falls dabei etwas schiefgeht, wird erneut zur Fehlermeldungsseite gesprungen.

```
mysql_query ("INSERT INTO gw_teilnahme VALUES ($id, $i, $antwort[$i])");
if (mysql_affected_rows() == 0) {
  header ("Location: error.html");
}
```

Unter dem PHP-Block befindet sich noch ein wenig HTML-Code. Hier wird lediglich eine kurze Teilnahmebestätigung angezeigt. PHP wird nur zur Anzeige des Benutzernamen, in der Überschrift verwendet:

```
<h1>Vielen Dank, <?php echo ($uname); ?>!</h1>
```

In Beispiel 3-2 sehen Sie den vollständigen Code des Listings, wiederum in klassischer mysql-Schreibweise. Alle für mysqli notwendigen Änderungen wurden im Prinzip bereits für das vorige beschrieben; auf der CD finden Sie natürlich trotzdem beide Versionen.

Beispiel 3-2: teilnahme.php – Benutzereingaben in die Datenbank schreiben, mysql-Version

```
<?php

  function cgi_param ($feld, $default="") {
    // Variable zunächst auf Default-Wert setzen
    $var = $default;
    if (isset($_GET[$feld]) && $_GET[$feld] != "") {
      // GET-Feld gefunden
      $var = $_GET[$feld];
    } elseif (isset($_POST[$feld]) && $_POST[$feld] != "") {
      // POST-Feld gefunden
      $var = $_POST[$feld];
    }
    // Ermittelten Wert zurückgeben
    return $var;
  }
```

Beispiel 3-2: teilnahme.php – Benutzereingaben in die Datenbank schreiben, mysql-Version (Fortsetzung)

```
// Datenbank-Verbindungsparameter
$host = "localhost";
$user = "winuser";
$pass = "G3w1nn3n";
$db   = "gewinnspiel";

// Vermutung: Alles korrekt ausgefüllt
$korrekt = 1;

// Formulardaten lesen
for ($i = 1; $i <= 4; $i++) {
  $antwort[$i] = cgi_param ("f$i", 0);
  if ($antwort[$i] == 0) {
    $korrekt = 0;
  }
}
$uname = cgi_param ("uname", "");
$email = cgi_param ("email", "");
$interest = cgi_param ("wish", 0);
if ($uname == "" || $email == "" || $interest == 0) {
  $korrekt = 0;
}

// Etwas nicht ausgefüllt?
if (!$korrekt) {
  header ("Location: spiel.php?fehler=1");
} else {
  // Verbindung zum MySQL-Server herstellen
  $conn = mysql_connect ($host, $user, $pass);
  // Datenbank auswählen
  mysql_select_db ($db);

  // Infos in die Datenbank schreiben
  mysql_query ("INSERT INTO gw_teilnehmer (tn_uname, tn_email,
        tn_interest) VALUES (\"$uname\", \"$email\", $interest)");
  if (mysql_affected_rows() == 0) {
    mysql_close();
    header ("Location: error.html");
  } else {
    // Teilnehmer-ID ermitteln
    $query = mysql_query ("SELECT tn_id from gw_teilnehmer
             WHERE tn_email=\"$email\"");
    list ($id) = mysql_fetch_row ($query);
    for ($i = 1; $i <= 4; $i++) {
      mysql_query ("INSERT INTO gw_teilnahme VALUES
                   ($id, $i, $antwort[$i])");
      if (mysql_affected_rows() == 0) {
        mysql_close();
        header ("Location: error.html");
      }
    }
```

Beispiel 3-2: teilnahme.php – Benutzereingaben in die Datenbank schreiben, mysql-Version (Fortsetzung)

```
    }
    mysql_close();
  }

?>

<html>
<head>
<title>Gewinnspiel</title>
<meta http-equiv="Content-type" content="text/html;charset=iso-8859-1" />
</head>
<body>
<h1>Vielen Dank, <?php echo ($uname); ?>!</h1>
<p>Wir freuen uns, dass Sie an unserem Gewinnspiel teilgenommen haben.
<br />
Unter allen richtigen Einsendungen verlosen wir am 01.08.2007 die Reise.
<br />
<br />
Der Gewinner wird per E-Mail benachrichtigt.</p>
</body>
</html>
```

Die PDO-Version funktioniert erneut ein wenig anders. Unter anderem wird an einer bestimmten Stelle wieder ein zweites `PDO`-Objekt benötigt. Um diesen Vorgang zu erleichtern, wurde hier eine Funktion namens `pdo_conn()` geschrieben, die ein fertiges Verbindungsobjekt mit den gewünschten Parametern zurückliefert. Wie Sie sehen, enthält der `new PDO()`-Aufruf diesmal ein Array mit zusätzlichen Parametern. Diese bedeuten kurz gesagt Folgendes:

- `PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION` sorgt dafür, dass SQL-Fehler nicht kommentarlos übergangen werden, sondern eine `PDOException` auslösen, die dann mittels `try/catch` abgefangen wird.
- `PDO::ATTR_PERSISTENT => 1` macht die Datenbankverbindung persistent, das heißt, sie bleibt geöffnet und kann wiederverwendet werden. Dies verbessert die Performance größerer datenbankbasierter Webanwendungen erheblich.

In Kapitel 8 lernen Sie noch weitere optionale PDO-Verbindungsparameter kennen. Beispiel 3-3 zeigt den gesamten PHP-Block des Beispiels in der PDO-Variante:

Beispiel 3-3: teilnahme.php – Benutzereingabe in die Datenbank schreiben, PDO-Version

```
<?php

  function cgi_param ($feld, $default="") {
    // Variable zunächst auf Default-Wert setzen
    $var = $default;
    if (isset($_GET[$feld]) && $_GET[$feld] != "") {
      // GET-Feld gefunden
      $var = $_GET[$feld];
```

Beispiel 3-3: teilnahme.php – Benutzereingabe in die Datenbank schreiben, PDO-Version (Fortsetzung)

```
  } elseif (isset($_POST[$feld]) && $_POST[$feld] != "") {
    // POST-Feld gefunden
    $var = $_POST[$feld];
  }
  // Ermittelten Wert zurückgeben
  return $var;
}

function pdo_conn () {
  // Verbindungsparameter
  $host = "localhost";
  $user = "winuser";
  $pass = "G3w1nn3n";
  $db   = "gewinnspiel";
  try {
    $conn = new PDO ("mysql:host=$host; dbname=$db", $user, $pass,
                array (PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                       PDO::ATTR_PERSISTENT => 1));
  } catch (PDOException $e) {
    echo ("FEHLER: ".$e->getMessage());
    return null;
  }
  return $conn;
}

try {
  // Erste Verbindung zum MySQL-Server
  $conn = pdo_conn();

  // Vermutung: Alles korrekt ausgefüllt
  $korrekt = 1;

  // Formulardaten lesen
  for ($i = 1; $i <= 4; $i++) {
    $antwort[$i] = cgi_param ("f$i", 0);
    if ($antwort[$i] == 0) {
      $korrekt = 0;
    }
  }
  $uname = cgi_param ("uname", "");
  $email = cgi_param ("email", "");
  $interest = cgi_param ("wish", 0);
  if ($uname == "" || $email == "" || $interest == 0) {
    $korrekt = 0;
  }

  // Etwas nicht ausgefüllt?
  if (!$korrekt) {
    header ("Location: spiel.php?fehler=1");
  } else {
```

Beispiel 3-3: teilnahme.php – Benutzereingabe in die Datenbank schreiben, PDO-Version (Fortsetzung)

```
      // Infos in die Datenbank schreiben
      $rows = $conn->exec ("ANSERT INTO gw_teilnehmer
                          (tn_uname, tn_email, tn_interest) VALUES
                          (\"$uname\", \"$email\", $interest)");
      if ($rows == 0) {
        header ("Location: error.html");
      } else {
        // Teilnehmer-ID ermitteln
        $query = $conn->query ("SELECT tn_id from gw_teilnehmer
                 WHERE tn_email=\"$email\"");
        list ($id) = $query->fetch();
        for ($i = 1; $i <= 4; $i++) {
          $conn2 = pdo_conn();
          $rows = $conn2->exec ("INSERT INTO gw_teilnahme VALUES
                               ($id, $i, $antwort[$i])");
          if ($rows == 0) {
            header ("Location: error.html");
          }
          $conn2 = null;
        }
      }
    }
    $conn = null;
  } catch (PDOException $e) {
    echo ("Fehler: ".$e->getMessage());
  }

?>
```

Den Gewinner ziehen

Das letzte PHP-Skript ist, wie bereits erwähnt, dem Website-Betreiber vorbehalten. Weiter unten wird entsprechend erläutert, wie Sie es in einem geschützten Website-Verzeichnis ablegen können. Zuerst werden in einer HTML-Tabelle alle Teilnehmer aufgelistet, die richtig geantwortet haben. Darunter erscheint der Name eines zufällig gewählten Einsenders als Gewinner. Damit der Site-Administrator ihn sofort benachrichtigen kann, wird die E-Mail-Adresse als Link dargestellt.



In der nachfolgenden Beschreibung wird aus Platz- und Übersichtsgründen ausschließlich die mysql-Syntax verwendet, dafür werden die Listings in Beispiel 3-4 und 3-5 zur Abwechslung in mysqli- und PDO-Schreibweise abgedruckt.

Damit überhaupt überprüft werden kann, ob die jeweiligen Antworten der Teilnehmer richtig sind, werden in einem ersten Schritt die Nummern der korrekten Antworten aus der Tabelle *gw_fragen* ermittelt und in einem neuen Array namens

$korrekt abgelegt. Die Funktion array_push ($array, $wert) dient dazu, den angegebenen Wert am Ende des Arrays anzuhängen:

```
// Nummern der richtigen Antworten speichern
$fr_query = mysql_query
     ("SELECT fr_korrekt FROM gw_fragen ORDER BY fr_id ASC");
$korrekt = array();
while (list ($fr_korrekt) = mysql_fetch_row ($fr_query)) {
  array_push ($korrekt, $fr_korrekt);
}
```

An dieser Stelle wird, was ziemlich fahrlässig ist, auf jegliche Fehlerkontrolle verzichtet. Da es sich nicht um die Verarbeitung von Benutzereingaben, sondern um das Auslesen von Daten aus einer selbst erstellten Tabelle handelt, ist das Risiko kalkulierbar. In einer echten Praxisanwendung sollten Sie allerdings auch solche vermeintlich sicheren Stellen überprüfen, um eventuelle eigene Fehler abzufangen – insbesondere während der Entwicklungs- und Testphase.

Als Nächstes werden die Nummern sämtlicher Teilnehmer in einem weiteren Array namens $teilnehmer gespeichert, um ihre Antworten danach in einer Schleife überprüfen zu können:

```
$tn_query = mysql_query ("SELECT tn_id FROM gw_teilnehmer");
$teilnehmer = array();
while (list ($tn_id) = mysql_fetch_row ($tn_query)) {
  array_push ($teilnehmer, $tn_id);
}
```

Mithilfe der soeben erstellten Liste werden die Antworten sämtlicher Teilnehmer überprüft. Die Nummern derjenigen, die korrekt geantwortet haben, werden in einem weiteren Array mit der Bezeichnung $korr_teilnehmer gespeichert. Die Überprüfung erfolgt nach dem weiter oben beim Einlesen der Formulardaten demonstrierten Schema: Die Variable $richtig wird zunächst auf 1 gesetzt, womit angenommen wird, dass alles richtig sei. Sobald eine falsche Antwort gefunden wird, erhält die Variable den Wert 0. Hier der entsprechende Codeblock:

```
$korr_teilnehmer = array();
foreach ($teilnehmer as $tn_id) {
  // Vermutung: Alles richtig
  $richtig = 1;
  // Alle Antworten des Teilnehmers durchlaufen
  $tl_query = mysql_query
            ("SELECT tl_frag, tl_antw FROM gw_teilnahme
              WHERE tl_tln=$tn_id ORDER BY tl_frag ASC");
  for ($i = 0; list ($tl_frag, $tl_antw) =
     mysql_fetch_row ($tl_query); $i++) {
     // Falsche Antwort?
     if ($tl_antw != $korrekt[$i]) {
       // Also nicht alles richtig
       $richtig = 0;
     }
  }
```

Die beiden Schleifenkonstrukte sollten Sie sich etwas näher anschauen. Eine `foreach()`-Schleife durchläuft nacheinander alle Elemente eines Arrays und speichert den Wert des jeweiligen Array-Elements in der hinter `as` angegebenen Variablen. Im vorliegenden Fall werden die Elemente aus der Teilnehmerliste nacheinander in `$tn_id` bereitgestellt. Auch die `for()`-Schleife mag auf den ersten Blick etwas ungewöhnlich aussehen: Die Bedingung überprüft nicht die Zählervariable `$i`, sondern liest jeweils den nächsten Datensatz der Abfrage `$tl_query`. Das ist praktisch, weil das Auslesen der Daten und die Erhöhung von `$i` als (bei 0 beginnender) Fragenummer auf diese Weise automatisch gleichzeitig stattfinden. Wenn Ihnen diese Konstruktion seltsam vorkommt, können Sie auch folgendes Synonym verwenden:

```
$i = 0;
while (list ($tl_frag, $tl_antw) = mysql_fetch_row ($tl_query)) {
  // Falsche Antwort?
  if ($tl_antw != $korrekt[$i]) {
  // Also nicht alles richtig
    $richtig = 0;
  }
  // Index erhöhen
  $i++;
}
```

Nach der Überprüfung aller Antworten wird der entsprechende Benutzer in `$korr_teilnehmer` gespeichert, falls `$richtig` den Wert 1 behalten hat:

```
  // Alles richtig?
  if ($richtig) {
    array_push ($korr_teilnehmer, $tn_id);
  }
}
```

Der PHP-Block wird nun kurz geschlossen, um auf bequeme Weise den Kopf der HTML-Tabelle anzuzeigen. Danach folgt eine weitere PHP-Sequenz, in der zunächst eine Liste der Lieblingsstädte erstellt wird, die der späteren Ausgabe dient:

```
$staedte = array ("Paris", "London", "Istanbul", "Rom");
```

Dann werden die Datensätze der korrekt antwortenden Benutzer als Tabellenzeilen ausgegeben. Dazu wird wieder eine `foreach()`-Schleife verwendet, diesmal über alle Elemente von `$korr_teilnehmer`:

```
// Ausgabe aller korrekten Einsendungen
foreach ($korr_teilnehmer as $kt) {
  $kt_query = mysql_query ("SELECT tn_uname, tn_email, tn_interest
                          FROM gw_teilnehmer WHERE tn_id=$kt");
  list ($tn_uname, $tn_email, $tn_interest) =
        mysql_fetch_row ($kt_query);
  echo "<tr>\n";
  echo "<td>$tn_uname</td>\n";
  echo "<td>$tn_email</td>\n";
  echo "<td>".$staedte[$tn_interest - 1]."</td>\n";
  echo "</tr>\n";
}
```

Beachten Sie den Ausdruck `$tn_interest - 1`, der als Index auf das Array `$staedte` angewendet wird: Da in der Datenbank die Werte 1 bis 4 für die Städte gespeichert werden, muss 1 abgezogen werden, damit der Wertebereich zu dem Array mit den Indizes 0 bis 3 passt.

Zum Schluss muss noch der Gewinner »gezogen« werden. Wie Sie sich vorstellen können, gibt es in einer programmgesteuerten Maschine wie dem Computer keinen echten Zufall.[4] Deshalb werden komplexe mathematische Verfahren zur Berechnung unvorhersagbarer Zahlenfolgen eingesetzt, die sogenannten *Zufallsgeneratoren*. In der Regel basieren sie auf der fortgesetzten Division sehr großer Zahlen. Wie beinahe jede Programmiersprache enthält auch PHP eine solche Komponente. Die betreffende Funktion heißt `rand()`. Als Parameter erhält sie optional den Mindest- und Höchstwert, im vorliegenden Fall die Indexgrenzen des Arrays `$korr_teilnehmer`: 0 und die um 1 verminderte Länge des Arrays.

```
$gnummer = rand (0, sizeof ($korr_teilnehmer) - 1);
```

Zu guter Letzt wird der Teilnehmer mit dieser Nummer aus der Datenbank ausgewählt und angezeigt:

```
$gewinner = $korr_teilnehmer[$gnummer];
$gw_query = mysql_query ("SELECT tn_uname, tn_email
     FROM gw_teilnehmer WHERE tn_id=$gewinner");
list ($tn_uname, $tn_email) = mysql_fetch_row ($gw_query);
echo ("Gewonnen hat: <b>$tn_uname</b>
     (<a href=\"mailto:$tn_email\">$tn_email</a>)");
```

In Beispiel 3-4 sehen Sie das vollständige Listing dieser Anwendung. Wie bereits erwähnt, wird hier die mysqli-Syntax verwendet; die mysql-Variante wurde bereits in den Erläuterungen gezeigt.

Beispiel 3-4: auswert.php – Antworten überprüfen und Gewinner »ziehen«, mysqli-Version

```
<?php

  // Verbindungsparameter
  $host = "localhost";
  $user = "winuser";
  $pass = "G3w1nn3n";
  $db   = "gewinnspiel";

  // Verbindung herstellen und Datenbank wählen
  $conn = new mysqli ($host, $user, $pass, $db);

?>
```

4 Da Pseudozufallszahlen besonders für starke Kryptografie immer noch zu vorhersagbar sind, versucht man inzwischen, Computer mit Quellen echter Entropie zu füttern – zum Beispiel radioaktiven Zerfall oder das Rauschen einer Radiofrequenz. Besuchen Sie die Website *http://www.random.org*, wenn das Thema Sie interessiert.

Beispiel 3-4: auswert.php – Antworten überprüfen und Gewinner »ziehen«, mysqli-Version (Fortsetzung)

```
<html>
<head>
<title>Gewinnspiel - Auswertung</title>
</head>
<body>
<h1>Gewinnspiel - Auswertung</h1>
<?php
  // Nummern der richtigen Antworten speichern
  $fr_query = $conn->query
             ("SELECT fr_korrekt FROM gw_fragen ORDER BY fr_id ASC");
  $korrekt = array();
  while (list ($fr_korrekt) = $fr_query->fetch_row()) {
    array_push ($korrekt, $fr_korrekt);
  }

  // Alle Teilnehmer ermitteln
  $tn_query = $conn->query ("SELECT tn_id FROM gw_teilnehmer");
  $teilnehmer = array();
  while (list ($tn_id) = $tn_query->fetch_row()) {
    array_push ($teilnehmer, $tn_id);
  }

  // Für jeden Teilnehmer herausfinden, ob er alles richtig beantwortet hat
  $korr_teilnehmer = array();
  foreach ($teilnehmer as $tn_id) {
    // Vermutung: Alles richtig
    $richtig = 1;
    // Alle Antworten des Teilnehmers durchlaufen
    $tl_query = $conn->query ("SELECT tl_frag, tl_antw FROM
            gw_teilnahme WHERE tl_tln=$tn_id ORDER BY tl_frag ASC");
    for ($i = 0; list ($tl_frag, $tl_antw) =
      $tl_query->fetch_row(); $i++) {
      // Falsche Antwort?
      if ($tl_antw != $korrekt[$i]) {
        // Also nicht alles richtig
        $richtig = 0;
      }
    }

    // Alles richtig?
    if ($richtig) {
      array_push ($korr_teilnehmer, $tn_id);
    }
  }

?>
Folgende Teilnehmer haben alle Fragen richtig beantwortet:<br />
<br />
<table border="2" cellpadding="4">
<tr>
```

Beispiel 3-4: auswert.php – Antworten überprüfen und Gewinner »ziehen«, mysqli-Version (Fortsetzung)

```
<th>Teilnehmer</th>
<th>E-Mail</th>
<th>Lieblingsstadt</th>
</tr>
<?php

  // Die Lieblingsstädte
  $staedte = array ("Paris", "London", "Istanbul", "Rom");
  // Ausgabe aller korrekten Einsendungen
  foreach ($korr_teilnehmer as $kt) {
    $kt_query = $conn->query ("SELECT tn_uname, tn_email,
                tn_interest FROM gw_teilnehmer WHERE tn_id=$kt");
    list ($tn_uname, $tn_email, $tn_interest) = $kt_query->fetch_row();
    echo "<tr>\n";
    echo "<td>$tn_uname</td>\n";
    echo "<td>$tn_email</td>\n";
    echo "<td>".$staedte[$tn_interest - 1]."</td>\n";
    echo "</tr>\n";
  }

?>
</table>
<br />
<?php

  // Den Gewinner "ziehen"
  $gnummer = rand (0, sizeof ($korr_teilnehmer) - 1);
  $gewinner = $korr_teilnehmer[$gnummer];

  // und ausgeben
  $gw_query = $conn->query ("SELECT tn_uname, tn_email FROM
              gw_teilnehmer WHERE tn_id=$gewinner");
  list ($tn_uname, $tn_email) = $gw_query->fetch_row();
  echo ("Gewonnen hat: <b>$tn_uname</b>
        (<a href=\"mailto:$tn_email\">$tn_email</a>)");

  // Datenbankverbindung schließen
  $conn->close();

?>
</body>
</html>
```

In Beispiel 3-5 wird die PHP Data Objects-Version gezeigt. Die wesentlichen Schnittstellenunterschiede, die zu den diversen Änderungen geführt haben, kennen Sie bereits.

Beispiel 3-5: auswert.php – Antworten überprüfen und Gewinner »ziehen«, PDO-Version

```
<?php

  function pdo_conn () {
    // Verbindungsparameter
    $host = "localhost";
    $user = "winuser";
    $pass = "G3w1nn3n";
    $db   = "gewinnspiel";
    try {
      $conn = new PDO ("mysql:host=$host; dbname=$db", $user, $pass,
              array (PDO::ERRMODE_EXCEPTION => true,
                     PDO::ATTR_PERSISTENT => true));
    } catch (PDOException $e) {
      echo ("FEHLER: ".$e->getMessage());
      return null;
    }
    return $conn;
  }

  try {
    // Erste Verbindung herstellen
    $conn = pdo_conn();

?>
<html>
<head>
<title>Gewinnspiel - Auswertung</title>
</head>
<body>
<h1>Gewinnspiel - Auswertung</h1>
<?php
    // Nummern der richtigen Antworten speichern
    $fr_query = $conn->query
              ("SELECT fr_korrekt FROM gw_fragen ORDER BY fr_id ASC");
    $korrekt = array();
    while (list ($fr_korrekt) = $fr_query->fetch()) {
      array_push ($korrekt, $fr_korrekt);
    }

    // Alle Teilnehmer ermitteln
    $tn_query = $conn->query ("SELECT tn_id FROM gw_teilnehmer");
    $teilnehmer = array();
    while (list ($tn_id) = $tn_query->fetch()) {
      array_push ($teilnehmer, $tn_id);
    }

    // Für jeden Teilnehmer herausfinden,
    // ob er alles richtig beantwortet hat
    $korr_teilnehmer = array();
    foreach ($teilnehmer as $tn_id) {
      // Vermutung: Alles richtig
      $richtig = 1;
```

Beispiel 3-5: auswert.php – Antworten überprüfen und Gewinner »ziehen«, PDO-Version (Fortsetzung)

```
      // Alle Antworten des Teilnehmers durchlaufen
      $tl_query = $conn->query ("SELECT tl_frag, tl_antw FROM
              gw_teilnahme WHERE tl_tln=$tn_id ORDER BY tl_frag ASC");
      for ($i = 0; list ($tl_frag, $tl_antw) =
        $tl_query->fetch(); $i++) {
        // Falsche Antwort?
        if ($tl_antw != $korrekt[$i]) {
          // Also nicht alles richtig
          $richtig = 0;
        }
      }

      // Alles richtig?
      if ($richtig) {
        array_push ($korr_teilnehmer, $tn_id);
      }
    }

?>
Folgende Teilnehmer haben alle Fragen richtig beantwortet:<br />
<br />
<table border="2" cellpadding="4">
<tr>
<th>Teilnehmer</th>
<th>E-Mail</th>
<th>Lieblingsstadt</th>
</tr>
<?php

    // Die Lieblingsstädte
    $staedte = array ("Paris", "London", "Istanbul", "Rom");
    // Ausgabe aller korrekten Einsendungen
    foreach ($korr_teilnehmer as $kt) {
      // Zusätzliche Verbindung
      $conn2 = pdo_conn();
      $kt_query = $conn2->query ("SELECT tn_uname, tn_email,
             tn_interest FROM gw_teilnehmer WHERE tn_id=$kt");
      list ($tn_uname, $tn_email, $tn_interest) = $kt_query->fetch();
      $conn2 = null;
      echo "<tr>\n";
      echo "<td>$tn_uname</td>\n";
      echo "<td>$tn_email</td>\n";
      echo "<td>".$staedte[$tn_interest - 1]."</td>\n";
      echo "</tr>\n";
    }

?>
</table>
<br />
```

Beispiel 3-5: auswert.php – Antworten überprüfen und Gewinner »ziehen«, PDO-Version (Fortsetzung)

```
<?php

    // Den Gewinner "ziehen"
    $gnummer = rand (0, sizeof ($korr_teilnehmer) - 1);
    $gewinner = $korr_teilnehmer[$gnummer];

    // und ausgeben
    $gw_query = $conn->query ("SELECT tn_uname, tn_email FROM
                gw_teilnehmer WHERE tn_id=$gewinner");
    list ($tn_uname, $tn_email) = $gw_query->fetch();
    echo ("Gewonnen hat: <b>$tn_uname</b>
           (<a href=\"mailto:$tn_email\">$tn_email</a>)");

    // Datenbankverbindung schließen
    $conn = null;
  } catch (PDOException $e) {
    echo ("FEHLER: ".$e->getMessage());
  }

?>
</body>
</html>
```

Die Fehlerseite

Um mögliche Fehler bei den Datenbankoperationen abzufangen, wird die Fehlerseite *error.html* angezeigt. Es handelt sich um ein einfaches, statisches HTML-Dokument, dessen Code Sie in Beispiel 3-6 sehen können.

Beispiel 3-6: error.html – Ausgabe bei Datenbankfehlern

```
<html>
<head>
<title>Gewinnspiel - Verarbeitungsfehler</title>
</head>
<body>
<h1>Gewinnspiel</h1>
Leider ist bei der Verarbeitung Ihrer Antworten ein unerwarteter
Fehler aufgetreten.<br />

Wenn Sie m&ouml;chten, k&ouml;nnen Sie es
<a href="spiel.php">erneut versuchen</a>.<br />
<br />
Bei einem permanenten Fehler bitten wir um eine kurze
<a href="mailto:webmaster@test.local">Mitteilung</a>.
</body>
</html>
```

Veröffentlichung und Test

Nachdem alle Skripten fertiggestellt sind, geht es daran, sie zu testen. In diesem Abschnitt wird beschrieben, wie Sie sie im Webserver veröffentlichen und im Browser ausprobieren können.

Die Anwendung im Webserver bereitstellen

Bevor Sie die Skripten testen können, müssen Sie sie in das Website-Verzeichnis Ihres Webservers kopieren und einige Vorbereitungen treffen. Wo Sie das Site-Verzeichnis von Apache finden, wurde im vorigen Kapitel im Rahmen der Installation beschrieben. Die nachfolgenden Beschreibungen gehen beispielsweise davon aus, dass die Site unter Linux im Verzeichnis */usr/local/apache2/htdocs* und unter Windows in *C:\Programme\Apache Software Foundation\Apache2.2\htdocs* liegt.

Erstellen Sie unter *htdocs* zunächst ein neues Verzeichnis namens *contest*. Kopieren Sie die Dateien *spiel.php*, *teilnahme.php* und *error.html* hinein.

Das Skript *auswert.php*, das nur vom Webmaster und nicht von externen Benutzern aufgerufen werden darf, soll in einem passwortgeschützten Unterverzeichnis liegen. Erstellen Sie dazu innerhalb des Verzeichnisses *contest* ein Unterverzeichnis mit der Bezeichnung *admin*.

Falls Web- und Datenbankserver nicht auf Ihrem eigenen Rechner ausgeführt werden, sondern bei einem Webhosting-Provider, müssen Sie die Dateien per FTP in Ihren Webspace übertragen. Gängige FTP-Software ermöglicht es Ihnen, auch auf dem entfernten Host die entsprechende Verzeichnisstruktur zu erstellen.

Nun geht es darum, Zugriffe auf das Verzeichnis *admin* nur nach Eingabe der korrekten Kombination aus Benutzername und Passwort zuzulassen. Apache stellt eine Reihe von Modulen bereit, die Authentifizierung (Benutzeranmeldung) in unterschiedlichen Formen und Sicherheitsstufen ermöglichen. Hier wird die einfachste Variante gezeigt: Die Benutzerdaten werden in einfachen Textdateien auf dem Serverrechner gespeichert. Browser werden aufgefordert, die Eingaben der Benutzer im Klartext zu senden (*Basic*-Authentifizierung).

Als Nächstes muss eine Textdatei mit den Anmeldedaten erstellt werden. Solche Dateien gehören aus Sicherheitsgründen nicht in das Website-Verzeichnis, weil ein Angreifer sie dort über den Webserver auslesen könnte. Falls sich die Site innerhalb von gemietetem Webspace bei einem Hosting-Provider befindet, sind Sie möglicherweise gezwungen, diesen Rat zu missachten. Dieses Szenario wird weiter unten mitsamt geeigneten Sicherheitsmaßnahmen behandelt.

Für die Erzeugung von Authentifizierungstextdateien ist das mit Apache gelieferte Kommandozeilentool *htpasswd* zuständig. Es befindet sich im *bin*-Verzeichnis Ihrer Apache-Installation – unter Linux standardmäßig */usr/local/apache2/bin*, auf Win-

dows-Systemen *C:\Programme\Apache Software Foundation\Apache2.2\bin*. Die Datei soll für dieses Beispiel *.htadmin* heißen und in einem neuen Verzeichnis namens *userdata* im Apache-Verzeichnis liegen. Sie soll den Benutzer *admin* und das Passwort *Pr1v4t* enthalten. Gehen Sie auf einem Linux-System beispielsweise folgendermaßen vor:

```
# cd /usr/local/apache2
# mkdir userdata
# bin/htpasswd -c userdata/.htadmin admin
New password:
Re-type new password:
Adding password for user admin
```

Auf Windows-Rechnern gilt dagegen folgendes Schema:

```
> C:
> cd Programme\Apache Software Foundation\Apache2.2
> md userdata
> bin\htpasswd -c userdata/.htadmin admin
New password: ******
Re-type new password: ******
Adding password for user admin
```

Wie Sie sehen, zeigt Linux bei der Passworteingabe gar kein Feedback an, unter Windows werden dagegen Sternchen (***) angezeigt. Beachten Sie unter Windows zudem die Pfadtrennzeichen: \ für das Betriebssystem und / für Apache-Konfiguration und -Dienstprogramme. Die *htpasswd*-Option -c sorgt dafür, dass die angegebene Datei neu erstellt wird. Wenn Sie mehrere Benutzerdaten in ein und derselben Datei unterbringen möchten, müssen Sie die Option beim ersten Mal verwenden und danach weglassen, andernfalls wird die Datei immer wieder neu angelegt und enthält nur den jeweils letzten Benutzer. Das Passwort wird verschlüsselt in der Datei gespeichert – auf Unix-Systemen wie die Systempasswörter im *crypt*-Format. Das sieht für den hier angelegten Benutzer *admin* beispielsweise so aus:

```
admin:Rm7B3Ft6SmuDo
```

Auf Windows-Rechnern kommt dagegen MD5-Verschlüsselung zum Einsatz; hier sieht der Eintrag folgendermaßen aus:

```
admin:$apr1$jX1.....$aN/g7ECzAZcy7Jm3tRy4N.
```

Anschließend müssen Sie in der Apache-Konfigurationsdatei *httpd.conf* einen Eintrag wie den folgenden hinzufügen, um das Verzeichnis *admin* mit diesen Anmeldedaten zu schützen:

```
<Location /contest/admin>
  AuthType Basic
  AuthBasicProvider file
  AuthName "Gewinnspiel Admin"
  AuthUserFile userdata/.htadmin
  Require user admin
</Location>
```

AuthType gibt die Form der Anmeldung an – hier *Basic* (klartextbasiert) im Gegensatz zur verschlüsselten Variante *Digest*. AuthBasicProvider legt fest, dass Apache die Vergleichsdaten aus einer htpasswd-Textdatei lesen soll.[5] AuthName bezeichnet den Anmeldebereich (*Realm*), den der Browser in der Anmeldeaufforderung anzeigt. Mittels AuthUserFile wird die soeben erstellte Datei mit den Anmeldedaten eingebunden. Bei einer Apache-Standardinstallation können Sie den Pfad – wie hier – relativ zur ServerRoot angeben; sollte dies nicht funktionieren, müssen Sie den vollständigen absoluten Pfad angeben, etwa */usr/local/apache2/userdata/.htadmin*.

Nachdem Sie diese Änderungen durchgeführt haben, müssen Sie Apache neu starten (siehe voriges Kapitel).

Nun folgt, wie oben versprochen, noch eine kurze Information über die nötigen Änderungen, wenn Sie die passwortgeschützte Seite in Ihrem Webspace bei einem Hoster betreiben möchten. Manche Hosting-Provider bieten Ihnen die bequeme Möglichkeit, den Passwortschutz für ein bestimmtes Verzeichnis über ihre Administrations-Website einzurichten. Sollte dies bei Ihrem Hoster nicht der Fall sein, können Sie wie folgt vorgehen:

1. Erstellen Sie die Datei *.htadmin* mit den Benutzerdaten unmittelbar im Verzeichnis *contest/admin*. Bei billigeren Hosting-Tarifen haben Sie keinen Shell-Zugriff auf den Serverrechner; und selbst wenn Sie per SSH zugreifen können, steht *htpasswd* vielleicht nicht zur Verfügung. In solchen Fällen müssen Sie die Datei auf einem anderen Rechner (am besten mit identischer Apache-Version) erstellen und in das genannte Verzeichnis kopieren.
2. Zusätzlich wird im Verzeichnis *admin* eine lokale Konfigurationsdatei namens *.htaccess* benötigt. Sie enthält die oben erläuterten Authentifizierungseinstellungen und sperrt den öffentlichen Zugriff auf Dateien, die mit *.ht* beginnen. Insgesamt benötigt die Datei folgenden Inhalt:

```
AuthType Basic
AuthBasicProvider file
AuthName "Gewinnspiel Admin"
AuthUserFile .htadmin
Require user admin
<Files .ht*>
  Order deny,allow
  Deny from all
</Files>
```

5 Bei Apache-Versionen bis 2.0.x müssen Sie diese Zeile weglassen, weil die Authentifizierung früher anders organisiert war.

Die Anwendung testen

Nun befinden sich die Skripten an der richtigen Stelle, um im Webbrowser ausprobiert zu werden. Starten Sie also den Browser Ihrer Wahl und geben Sie die Adresse des Skripts *spiel.php* ein. Sollten Sie der Konfigurationsanleitung bis hierher gefolgt sein, lautet die URL *http://www.test.local/contest/spiel.php* oder – auf dem Serverrechner selbst – *http://127.0.0.1/contest/spiel.php*. Wenn Sie alles richtig gemacht haben, müsste das Spielformular so aussehen wie in Abbildung 3-2. Kreuzen Sie nun die Antworten an und geben Sie beliebige Benutzerdaten ein. Die nachfolgende

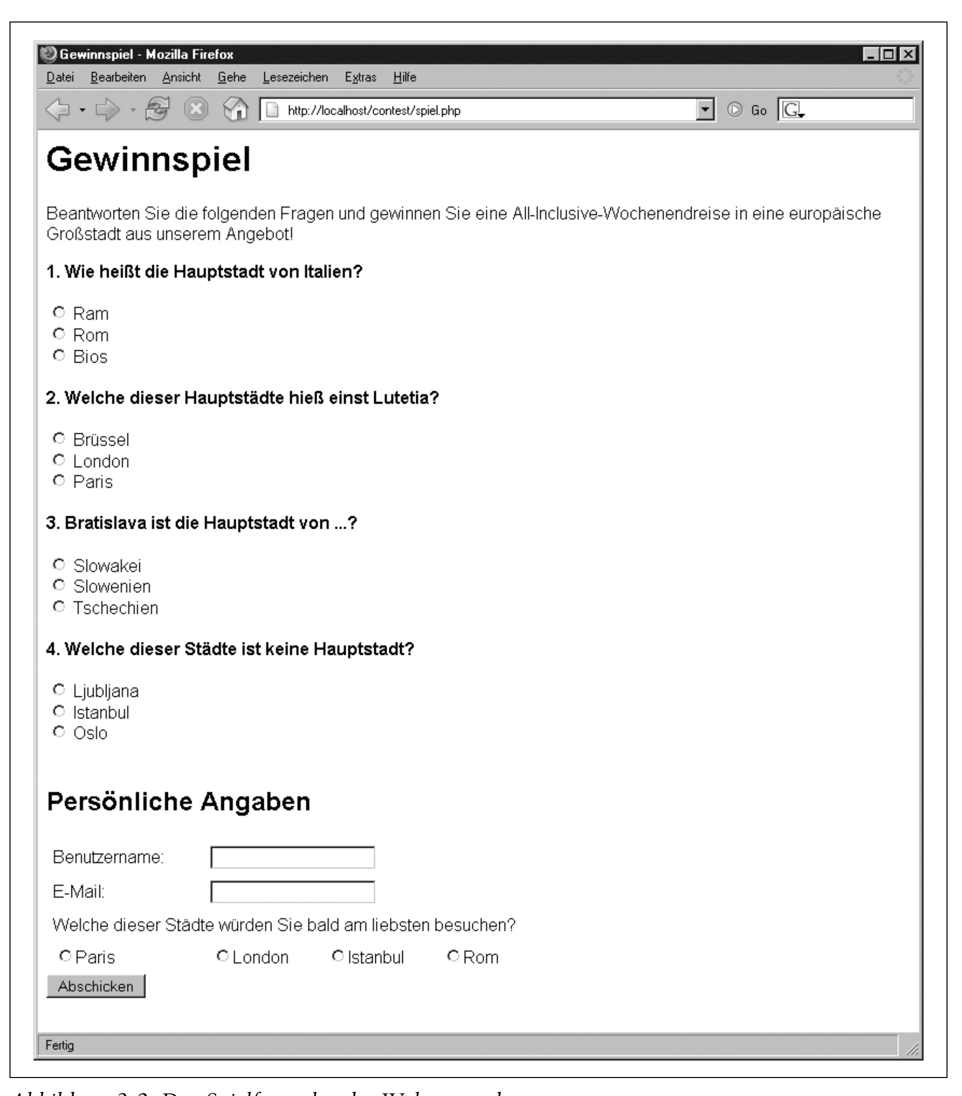


Abbildung 3-2: Das Spielformular der Webanwendung

Beschreibung geht davon aus, dass Sie das Skript dreimal ausführen und dabei folgende Eingaben tätigen (zweimal richtig, einmal falsch):

- Alle Antworten richtig (*Rom*, *Paris*, *Slowakei*, *Istanbul*); Benutzername *Klaus*; E-Mail *klaus@example.com*; Wunschstadt *London*.
- Alle Antworten richtig (siehe oben); Benutzername *Sabine*; E-Mail *sabine@example.net*; Wunschstadt *Instanbul*.
- Zwei Antworten richtig und zwei falsch (*Rom*, *London*, *Slowenien*, *Istanbul*); Benutzername *Hans*; E-Mail *hans@example.org*; Wunschstadt *Rom*.

Nach diesen Operationen hat die Datenbanktabelle *gw_teilnehmer* folgende Inhalte:

```
+-------+----------+-------------------+-------------+
| tn_id | tn_uname | tn_email          | tn_interest |
+-------+----------+-------------------+-------------+
|     1 | Klaus    | klaus@example.com  | 2           |
|     2 | Sabine   | sabine@example.net | 3           |
|     3 | Hans     | hans@example.org   | 4           |
+-------+----------+-------------------+-------------+
```

In der Tabelle *gw_teilnahme* finden Sie dagegen die folgenden Datensätze:

```
+--------+---------+---------+
| tl_tln | tl_frag | tl_antw |
+--------+---------+---------+
|      1 |       1 |       2 |
|      1 |       2 |       3 |
|      1 |       3 |       1 |
|      1 |       4 |       2 |
|      2 |       1 |       2 |
|      2 |       2 |       3 |
|      2 |       3 |       1 |
|      2 |       4 |       2 |
|      3 |       1 |       2 |
|      3 |       2 |       2 |
|      3 |       3 |       2 |
|      3 |       4 |       2 |
+--------+---------+---------+
```

Jedes Mal, wenn Sie in *spiel.php* die Schaltfläche *Abschicken* anklicken, erscheint die Ausgabe von *teilnahme.php*. In Abbildung 3-3 sehen Sie ein Beispiel für den Teilnehmer Klaus. Nun sind genügend Testdaten vorhanden, um das Administrationsskript *auswert.php* ausprobieren zu können. Geben Sie also dessen URL (z.B. *http://127.0.0.1/contest/admin/auswert.php*) in die Adresszeile des Browsers ein. Zunächst werden Sie aufgefordert, den Benutzernamen und das Passwort einzugeben. Abbildung 3-4 zeigt, wie das im Browser Firefox aussieht.

In Abbildung 3-5 sehen Sie schließlich die Beispielausgabe von *auswert.php*. Wenn Sie die Seite einige Male neu laden, werden Sie feststellen, dass die beiden Benutzer, die richtig geantwortet haben, auf lange Sicht etwa gleich oft gewinnen.

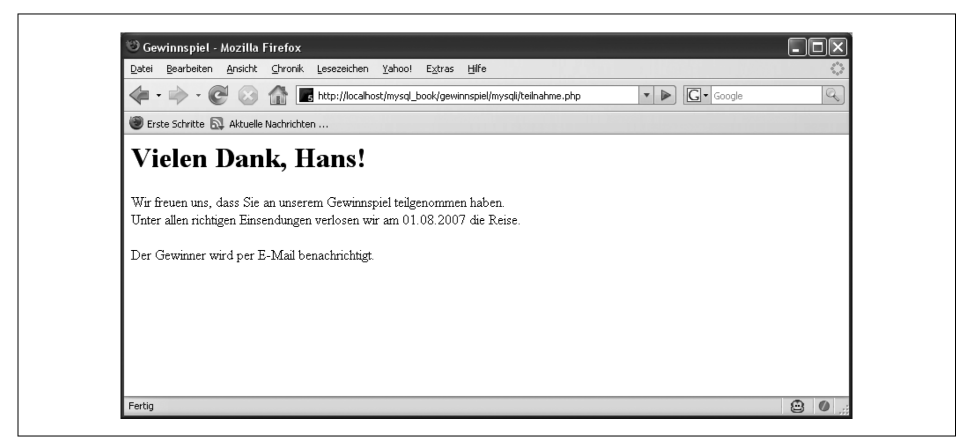


Abbildung 3-3: Die Dankeschön-Seite der Webanwendung

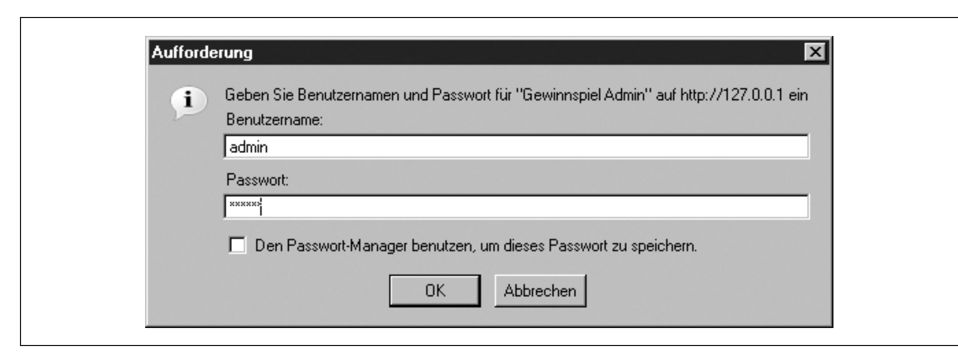


Abbildung 3-4: Benutzeranmeldung für das Auswertungsskript



Abbildung 3-5: Ermittlung des Gewinners aus allen richtigen Einsendungen

In diesem Kapitel:

- Der Kommandozeilenclient mysql
- phpMyAdmin

KAPITEL 4

Mit MySQL arbeiten

Arbeit um der Arbeit willen ist gegen die menschliche Natur.

John Locke

Nach dem praktischen Schnelleinstieg im vorigen Kapitel erfahren Sie in den folgenden Kapiteln ausführlich, wie Sie MySQL-Datenbanken erstellen und bearbeiten können. Zunächst werden in diesem Kapitel zwei wichtige Tools für die Arbeit mit MySQL genauer behandelt: der Kommandozeilenclient mysql und der Webclient phpMyAdmin.

Der Kommandozeilenclient mysql

In den beiden vorangehenden Kapiteln haben Sie bereits intuitiv mit dem Konsolentool mysql gearbeitet. An dieser Stelle werden seine wichtigsten Funktionen systematischer vorgestellt.

Einführung

Wie Sie bereits erfahren haben, wird das Programm üblicherweise mit den Optionen `-u` *<Benutzername>* und `-p` (ohne Wert) für die anschließende Passworteingabe gestartet, zum Beispiel:

```
> mysql -u root -p
Enter password: *******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.0.41-community-nt MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
```

Die Sternchen für die Passworteingabe werden nur unter Windows angezeigt; auf Unix-Rechnern erfolgt wie üblich kein Feedback.

Es gibt zahlreiche weitere Optionen für den Start von `mysql`. Wenn Sie eine Gesamtübersicht erhalten möchten, können Sie Folgendes eingeben:

```
$ mysql --help
```

Da die Ausgabe zu lang ist, sollten Sie sie durch eine Pipe an einen Pager wie *less* (Linux) oder *more* (Windows) weiterleiten:

```
$ mysql --help |less
> mysql --help |more
```

Alternativ können Sie den gesamten Text auf beiden Systemen in eine Datei umleiten:

```
$ mysql --help >myhelp.txt
```

Hier noch zwei weitere wichtige Kommandozeilenoptionen:

`-h=Hostname` *oder* `--host=Hostname`

Verbindung zum MySQL-Server auf einem anderen Rechner herstellen. Sie können einen Hostnamen wie *dbserver.test.local* oder eine IP-Adresse wie *192.168.0.17* angeben.

`--tee=Dateiname`

Sämtliche Ein- und Ausgaben der `mysql`-Sitzung zusätzlich in die angegebene Datei schreiben. Falls die Datei bereits besteht, werden die neuen Daten angehängt. Dieselbe Funktion können Sie auch mithilfe eines weiter unten gezeigten Clientbefehls ein- und wieder ausschalten.

Hinter sämtlichen Optionen können Sie den Namen der Datenbank angeben, die als Standard verwendet werden soll. Diese kann innerhalb des Programms mittels `use` (siehe unten) geändert werden.

Das folgende Beispiel stellt eine Verbindung mit dem MySQL-Server auf dem Host *db.test.local* her, kopiert die Ein- und Ausgaben in die Datei *mysql.log* und wählt die Datenbank *gewinnspiel* als Standard aus:

```
$ mysql -u root -p -h=db.test.local --tee=mysql.log gewinnspiel
```

Nach erfolgreichem Login erscheint der Prompt des Clients:

```
mysql>
```

Nach dem Start sollten Sie sich als Erstes um die bereits in Kapitel 2 angedeutete Zeichensatzproblematik kümmern. Die meisten modernen Unix-Terminals verwenden standardmäßig utf-8, während die Windows-Eingabeaufforderung aus historischen Gründen den MS-DOS-Zeichensatz benutzt. MySQL selbst setzt dagegen von Hause aus ISO-Latin-1 (auch als ISO-8859-1 bekannt) ein.

Damit Umlaute, Sonderzeichen und andere nicht lateinische Zeichen korrekt zwischen dem MySQL-Server und dem Client ausgetauscht werden, müssen Sie diese Präferenzen einander anpassen. Grundsätzlich gibt es zwei Möglichkeiten dafür:

Entweder Sie ändern den Terminal-Zeichensatz in ISO-Latin-1, oder Sie passen den Zeichensatz, über den der Client selbst mit dem Server kommuniziert an. Beachten Sie, dass beides nichts mit dem konkreten Zeichensatz der jeweiligen Datenbank, Tabelle oder gar einzelnen Spalte zu tun hat – diese Umsetzung erledigt MySQL selbstständig.

Der Zeichensatz eines Unix-Terminalfensters wird nach System und Terminalanwendung unterschiedlich geändert; lesen Sie gegebenenfalls die Dokumentation Ihrer konkreten Benutzeroberfläche. Hier nur zwei der verbreitetsten Terminals als Beispiel: In der KDE-Konsole wird der Menübefehl *Einstellung → Kodierung → Westeuropäisch (iso 8859-1)* verwendet. Im GNOME-Terminal lautet der Menüpunkt dagegen *Terminal → Zeichenkodierung festlegen → Westlich (ISO-8859-15)*. ISO-8859-15 ist übrigens eine modernere Variante von Latin-1 – der einzige Unterschied besteht darin, dass das Eurozeichen (€) hinzugefügt wurde.

Wenn Sie den voreingestellten Zeichensatz der Windows-Eingabeaufforderung ändern möchten, können Sie folgenden Befehl eingeben:

```
> chcp 1252
```

Windows-1252 ist nicht ganz, aber fast identisch mit Latin-1. Nun müssen Sie noch zusätzlich die Titelleiste des Eingabeaufforderungsfensters mit der rechten Maustaste anklicken und auf der Registerkarte *Schriftart* die Schrift *Lucida Console* wählen, da die alte DOS-Rasterschriftart nicht mit dem Windows-Zeichensatz kompatibel ist.

Einfacher als die Änderung des Konsolen-Zeichensatzes ist oft die Anpassung des Zeichensatzes, über den der Client mit dem MySQL-Server kommuniziert. Dazu wird der MySQL-Befehl `SET NAMES` verwendet. In der Windows-Eingabeaufforderung wird dann Folgendes eingegeben:

```
mysql> SET NAMES cp850;
```

In den meisten Unix-Terminals, die UTF-8 einsetzen, lautet das passende Kommando dagegen:

```
mysql> SET NAMES utf8;
```

Nach diesen Vorbereitungen können Sie beliebige SQL-Abfragen sowie interne Befehle des Clients selbst eingeben. Für eine erste Übersicht über Letztere können Sie Folgendes eingeben:

```
mysql> \h
```

Für jeden internen Befehl gibt es eine Abkürzung mit vorangestelltem Backslash (\). `\h` ist beispielsweise die Kurzfassung für `help`. Ein weiteres Synonym für `help` ist übrigens das Fragezeichen – sowohl mit (`\?`) als auch ohne Backslash (`?`). Abbildung 4-1 zeigt den Start des Clients und die Ausgabe des `help`-Befehls unter openSUSE 10.2. In Abbildung 4-2 wird dasselbe auf einem Windows-System gezeigt; wie Sie sehen, stehen dort etwas weniger Optionen zur Verfügung.

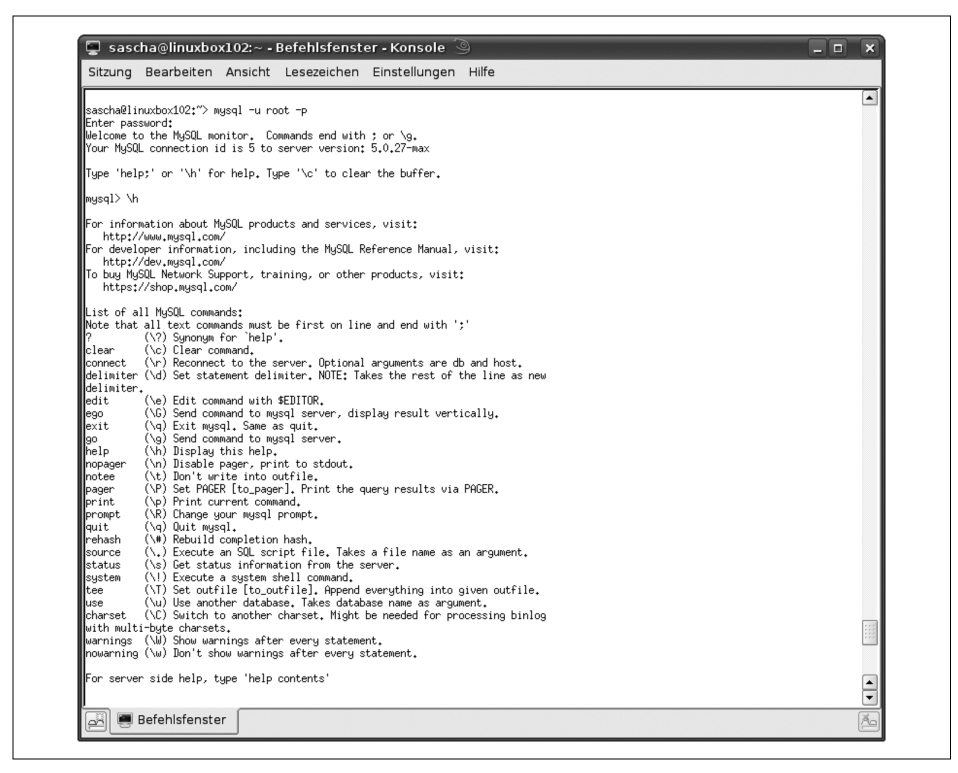


Abbildung 4-1: Der Kommandozeilenclient mysql und seine Hilfe unter Linux

```
Eingabeaufforderung - mysql -u root -p
C:\Dokumente und Einstellungen\Sascha>mysql -u root -p
Enter password: *******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.1.15-beta-community-nt MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> \h

For information about MySQL products and services, visit:
   http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
   http://dev.mysql.com/
To buy MySQL Network Support, training, or other products, visit:
   https://shop.mysql.com/

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear command.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter. NOTE: Takes the rest of the line as new delimiter.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
notee     (\t) Don't write into outfile.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
tee       (\T) Set outfile [to_outfile]. Append everything into given outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing binlog with multi-byte char
sets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.

For server side help, type 'help contents'
```

Abbildung 4-2: Der Kommandozeilenclient mysql und seine Hilfe unter Windows

Mit Ausnahme der Kurzbefehle muss jede Eingabe – ob interner Befehl oder SQL-Abfrage – durch ein Semikolon abgeschlossen werden. Dies erlaubt mehrere Anweisungen in einer einzelnen Zeile. Das folgende Beispiel wählt zunächst die Datenbank *gewinnspiel* als Standard aus und zeigt anschließend die Datensätze der Tabelle *gw_fragen* an:

```
mysql> use gewinnspiel; SELECT * FROM gw_fragen;
```

Auch die Verteilung langer Eingaben auf beliebig viele Zeilen wird unterstützt. Das nachfolgende Beispiel wählt die Fragetexte und die jeweils richtige Antwort für das Gewinnspiel aus:

```
mysql> SELECT fr_frage, an_text
    -> FROM gw_fragen, gw_antworten
    -> WHERE fr_id=an_frage AND fr_korrekt=an_antwort;
+-----------------------------------------------+----------+
| fr_frage                                      | an_text  |
+-----------------------------------------------+----------+
| Wie heißt die Hauptstadt von Italien?         | Rom      |
| Welche dieser Hauptstädte hieß einst Lutetia? | Paris    |
| Bratislava ist die Hauptstadt von ...?        | Slowakei |
| Welche dieser Städte ist keine Hauptstadt?    | Istanbul |
+-----------------------------------------------+----------+
4 rows in set (0.00 sec)
```

Statt des abschließenden Semikolons können Sie auch die Zeichenfolge `\g` (den Befehl go) verwenden. Die spezielle Variante `\G` (ego) sorgt dagegen für eine andere Ausgabe: Jedes Feld wird in einer separaten Zeile ausgegeben. Das ist nützlich, wenn die Spalten einer Abfrage zu breit sind, so dass die Tabelle nicht korrekt dargestellt werden kann. Hier die obige Abfrage in diesem Format:

```
mysql> SELECT fr_frage, an_text
    -> FROM gw_fragen, gw_antworten
    -> WHERE fr_id=an_frage AND fr_korrekt=an_antwort \G
*************************** 1. row ***************************
fr_frage: Wie heißt die Hauptstadt von Italien?
 an_text: Rom
*************************** 2. row ***************************
fr_frage: Welche dieser Hauptstädte hieß einst Lutetia?
 an_text: Paris
*************************** 3. row ***************************
fr_frage: Bratislava ist die Hauptstadt von ...?
 an_text: Slowakei
*************************** 4. row ***************************
fr_frage: Welche dieser Städte ist keine Hauptstadt?
 an_text: Istanbul
4 rows in set (0.00 sec)
```

Auf Wunsch können Sie das Semikolon auch durch ein anderes Zeichen ersetzen. Erforderlich wird dies zum Beispiel bei der manuellen Eingabe von Stored Procedures und Triggern, da diese Semikola enthalten (siehe Kapitel 7). Verwenden Sie hier-

für den Befehl `DELIMITER` oder `\d`. In der Langform müssen Sie dazu das bisherige Schlusszeichen anhängen – das folgende Beispiel stellt den Doppelpunkt (`:`) ein:

```
mysql> delimiter :;
```

Zurück zum Semikolon geht es danach wie folgt:

```
mysql> delimiter ;:
```

Die Kurzfassung `\d` macht diese Einstellungen übersichtlicher und verständlicher:

```
mysql> \d :
mysql> \d ;
```

Ein weiteres besonderes Kürzel ist `\c` (`clear`): Es ermöglicht den Abbruch einer bereits begonnenen, insbesondere mehrzeiligen Eingabe, zum Beispiel:

```
mysql> SELECT * FROM gw_fragen
    -> WHERE \c
```

Beachten Sie, dass dies innerhalb von Anführungszeichen nicht funktioniert. Sie müssen diese zuerst schließen. Das folgende Beispiel fügt einen längeren Text in eine (fiktive) Tabelle ein und versucht dann, die Eingabe abzubrechen. Dies funktioniert erst beim zweiten Anlauf, nachdem die Anführungszeichen geschlossen wurden:

```
mysql> INSERT INTO test (beschreibung) VALUES "Dieser Text
    "> geht über mehrere Zeilen\c
    "> hinweg."
    -> \c
```



Die Langfassungen interner Befehle, die nicht für sich stehen, sondern andere Befehle ergänzen oder manipulieren, können Sie nur dann verwenden, wenn Sie den Client mit der speziellen Option `--named-commands` gestartet haben. Ein Beispiel wäre ein ausgeschriebenes `clear` statt `\c`.

Einen der wichtigsten internen Befehle des Clients haben Sie bereits kennengelernt: `use` *`Datenbank`* (Kurzfassung `\u` *`Datenbank`*) wählt die angegebene Datenbank aus, was erheblich komfortabler ist, als in jeder Abfrage *`Datenbank.Tabellenname`* schreiben zu müssen. Wenn in derselben Zeile kein weiterer Befehl steht, kommt `use` auch ohne Semikolon aus. Beispiel:

```
mysql> use gewinnspiel
Database changed
```

Möchten Sie herausfinden, welche Datenbank gerade Standard ist, können Sie per `SELECT`-Abfrage den Wert der Funktion `DATABASE()` ermitteln:

```
mysql> SELECT DATABASE();
+-------------+
| DATABASE()  |
+-------------+
```

```
| gewinnspiel |
+-------------+
1 row in set (0.01 sec)
```

Sollten Sie keine spezielle Datenbank ausgewählt haben, erhalten Sie die Ausgabe `NULL`.

Die SQL-Anweisung `SHOW` ermöglicht die Anzeige weiterer Informationen. `SHOW DATABASES` gibt alle auf dem Server verfügbaren Datenbanken aus, zum Beispiel:

```
mysql> SHOW DATABASES;
+-------------+
| Database    |
+-------------+
| buchliste   |
| essen       |
| fachbuecher |
| gewinnspiel |
| mysql       |
| reisebuero  |
| test        |
+-------------+
17 rows in set (0.02 sec)
```

`SHOW TABLES` zeigt dagegen die Tabellen der aktuellen Datenbank an – natürlich nur, sofern eine Standarddatenbank ausgewählt wurde:

```
mysql> SHOW TABLES;
+-----------------------+
| Tables_in_gewinnspiel |
+-----------------------+
| gw_antworten          |
| gw_fragen             |
| gw_teilnahme          |
| gw_teilnehmer         |
+-----------------------+
4 rows in set (0.00 sec)
```

Auch die Struktur einer Tabelle können Sie anzeigen lassen. Die zuständige Abfrage besitzt die Syntax `DESCRIBE [`*`Datenbank`*`.]`*`Tabelle`* (oder kurz `DESC`). Den Datenbanknamen benötigen Sie nur, falls sich die gewünschte Tabelle nicht in der aktuellen Datenbank befindet. Hier sehen Sie ein Beispiel:

```
mysql> DESCRIBE gw_fragen;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| fr_id      | int(11)     |      | PRI | NULL    | auto_increment |
| fr_frage   | varchar(80) | YES  |     | NULL    |                |
| fr_korrekt | int(11)     | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

Weitere Optionen

Mithilfe des Befehls `tee` oder `\T` können Sie auch im laufenden Betrieb eine Datei festlegen, an die sämtliche Befehlseingaben und Ergebnisse angehängt werden, zum Beispiel:

```
mysql> tee mysql.log;
```

Wenn Sie die Dialoge nicht mehr in eine Datei kopieren möchten, können Sie den Befehl `notee` oder `\t` verwenden:

```
mysql> notee;
```

Sehr interessant ist auch der Befehl `source` *`Dateiname`* (Kurzfassung `\.`). Er dient dazu, SQL-Abfragen aus einer externen (Text-)Datei zu lesen und auszuführen. Das folgende Beispiel bezieht sich auf die Erstellung der ab dem nächsten Kapitel benötigten Datenbank *reisebuero*:

```
mysql> \. reisebuero.sql
```

Die Datei *reisebuero.sql* finden Sie übrigens im Verzeichnis *databases* auf der beiliegenden CD-ROM.

Alternativ können Sie SQL-Quelldateien übrigens auch per Eingabeumleitung an den `mysql`-Client übergeben, wenn er noch nicht läuft:

```
$ mysql -u root -p <reisebuero.sql
Enter password:
```

Speziell für solche Quelldateien ist es übrigens nützlich, jede einzelne Abfrage mit `\p` (Print) vor dem Semikolon abzuschließen – dadurch werden nicht nur Ergebnisse wie `Database created`, sondern auch die Abfragen selbst angezeigt. Das folgende Beispiel ist die erste Zeile von *reisebuero.sql* – sie löscht diese Datenbank, falls sie bereits vorhanden sein sollte:

```
mysql> DROP DATABASE IF EXISTS reisebuero \p;
```

SQL-Dateien als »Futter« für `source` können Sie mithilfe des Kommandozeilentools *mysqldump* erstellen. Auf diese Weise lassen sich existierende Datenbanken leicht exportieren. Dies wird in Kapitel 9 ausführlicher erläutert.

`status` oder `\s` liefert Statusinformationen über MySQL-Server und -Client. Ein Beispiel:

```
mysql> \s
--------------
mysql  Ver 14.13 Distrib 5.1.18-beta, for Win32 (ia32)

Connection id:          2
Current database:       gewinnspiel
Current user:           root@localhost
SSL:                    Not in use
Using delimiter:        ;
```

```
Server version:         5.1.18-beta-community-nt MySQL Community Server (GPL)
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:    latin1
Conn.  characterset:    latin1
TCP port:               3306
Uptime:                 7 hours 11 min 24 sec

Threads: 1  Questions: 28  Slow queries: 0  Opens: 19
Flush tables: 1  Open tables: 0  Queries per second avg: %
--------------
```

Unter Unix unterstützt der mysql-Client die Eingabevervollständigung genau so wie die Shell. Wenn Sie nach einigen Zeichen eines Datenbank-, Tabellen- oder Spaltennamens Tab drücken, wird dieser Name ergänzt, soweit dies eindeutig möglich ist. Angenommen, Sie arbeiten mit der Datenbank *gewinnspiel* aus dem vorigen Kapitel. Geben Sie zum Beispiel

```
mysql> SELECT * FROM gw_t
```

ein und betätigen Sie dann die Tab-Taste. Da die beiden Tabellen *gw_teilnehmer* und *gw_teilnahme* mit `gw_t` beginnen, wird der Text zu `gw_teiln` ergänzt. Nun können Sie beispielsweise ein `e` oder `a` eingeben und nochmals Tab drücken, um den Tabellennamen fertigzustellen. Alternativ können Sie auch an jeder Stelle zweimal hintereinander Tab drücken, um eine Liste aller verfügbaren Möglichkeiten (Tabellen und enthaltener Spalten) zu generieren. Beispiel:

```
mysql> SELECT * from gw_teiln Tab Tab
gw_teilnahme              gw_teilnehmer.tn_email
gw_teilnahme.tl_antw      gw_teilnehmer.tn_id
gw_teilnahme.tl_frag      gw_teilnehmer.tn_interest
gw_teilnahme.tl_tln       gw_teilnehmer.tn_uname
gw_teilnehmer
mysql> SELECT * from gw_teiln
```

Eine weitere interessante Option besteht darin, den Prompt des Clients zu ändern. Standardmäßig lautet er bekanntlich `mysql>`. Der Befehl zur Modifikation heißt `\R` oder in der Langform `prompt`. Der gewünschte Text wird ohne Anführungszeichen dahinter gesetzt; wenn Sie ein Leerzeichen am Ende haben möchten, müssen Sie es auch hinschreiben. Das folgende Beispiel setzt den Prompt auf `#` und danach wieder zurück auf `mysql>`:

```
mysql> \R #
PROMPT set to '# '
# \R mysql>
PROMPT set to 'mysql> '
mysql>
```

Neben beliebigen Zeichen können Sie auch eine Reihe spezieller Platzhalter einsetzen. Das folgende Beispiel stellt den Hostnamen (`\h`), einen Doppelpunkt, den Namen der aktuellen Standarddatenbank (`\d`) und ein Größer-Zeichen ein:

```
mysql> \R \h:\d>
localhost:gewinnspiel>
```

In Tabelle 4-1 sehen Sie eine Übersicht über alle verfügbaren Platzhalter.

Tabelle 4-1: Platzhalter für den Prompt des Kommandozeilenclients

Platzhalter	Bedeutung
`\v`	Serverversion
`\d`	aktuelle Standarddatenbank
`\h`	Serverhost
`\p`	TCP-Port oder Unix-Domain-Socket
`\u`	Benutzername
`\U`	Benutzerkonto (User@Host)
`\\`	Backslash als Zeichen (\)
`\n`	Zeilenumbruch
`\t`	Tabulator
`\ `	Leerzeichen (Space)
`\_`	Leerzeichen
`\R`	Stunde der aktuellen Uhrzeit (0–23)
`\r`	Stunde der aktuellen Uhrzeit (0–12)
`\m`	Minuten der aktuellen Uhrzeit
`\y`	aktuelles Jahr, zweistellig
`\Y`	aktuelles Jahr, vierstellig
`\D`	aktuelles Datum
`\s`	Sekunden der aktuellen Uhrzeit
`\w`	aktueller Wochentag, 3 Buchstaben (Mon, Tue, ...)
`\P`	am/pm
`\o`	aktueller Monat, nummerisch
`\O`	aktueller Monat, 3 Buchstaben (Jan, Feb, ...)
`\c`	Zähler, der bei jedem Befehl erhöht wird
`\l`	aktueller Delimiter (seit 5.0.25)
`\S`	Semikolon
`\'`	einfaches Anführungszeichen
`\"`	doppeltes Anführungszeichen

Um Ihre Arbeit mit dem Client zu beenden, können Sie zu guter Letzt `exit` oder `quit` (Kurzfassung jeweils `\q`) eingeben:

```
mysql> \q
Bye
```

Auf Unix-Systemen funktioniert alternativ auch die Tastenkombination Strg + D zum Beenden des Clients.

Befehlsübersicht

Tabelle 4-2 zeigt eine Übersicht über sämtliche internen Befehle des Kommandozeilenclients – sowohl die bereits besprochenen als auch einige weniger häufig genutzte. Die mit einem Sternchen (*) gekennzeichneten sind nur auf Unix-Rechnern verfügbar. Wenn die Langfassung in Klammern steht, kann sie nur nach einem Start mit `--named-commands` verwendet werden.

Tabelle 4-2: Die internen Kommandos des Kommandozeilenclients mysql im Überblick

Kommando	Langfassung	Bedeutung
`\h, \?, ?`	`help`	Liste aller internen Befehle anzeigen.
`\c`	`(clear)`	Aktuelle Eingabe ignorieren (bei Fehlern).
`\r`	`(reconnect)`	Neue Verbindung zum MySQL-Server mit denselben Anmeldedaten herstellen.
`\d`	`delimiter`	Abschlusszeichen für Befehlseingabe ändern (Standard: `;`).
`\e *`	`edit`	Befehl mit dem Editor aus der Umgebungsvariablen `$EDITOR` (meist vi) bearbeiten. Durch Beenden des Editors mit Speichern (bei vi durch Eingabe von `:wq`) wird die Eingabe übernommen.
`\G`	`(ego)`	Abfrageergebnis zeilen- statt spaltenweise anzeigen.
`\q`	`exit, quit`	Client beenden.
`\g`	`(go)`	Befehl abschicken (funktioniert unabhängig vom eingestellten Delimiter immer).
`\t`	`notee`	Protokollierung in Textdatei beenden.
`\P *`	`pager`	Lange Ergebnisse mittels `$PAGER` (meist less) seitenweise ausgeben.
`\p`	`(print)`	Aktuellen Befehl ausgeben (praktisch für Feedback in SQL-Dateien).
`\R`	`prompt`	Eingabeaufforderung ändern (Standard: `mysql>`).
`\# *`	`rehash`	Tabellen- und Spaltennamen für die Eingabevervollständigung nach Änderungen neu einlesen.
`\.`	`source`	Kommandos aus der angegebenen Datei ausführen.
`\s`	`status`	Statusinformationen anzeigen.
`\T`	`tee`	Protokollierung der gesamten Ein- und Ausgabe in die angegebene Datei.
`\u`	`use`	Standarddatenbank wechseln.
`\C`	`charset`	Zeichensatz ändern (nicht zu verwechseln mit `SET NAMES`).
`\W`	`warnings`	Warnungen automatisch anzeigen (ansonsten nur einzeln durch `SHOW WARNINGS;`).
`\w`	`nowarning`	Warnungen nicht mehr automatisch anzeigen.

phpMyAdmin

In Kapitel 2 wurde bereits beschrieben, wie Sie den webbasierten Client *phpMyAdmin* einrichten und starten können. Wenn Sie diese Installationsanleitung befolgt haben, müsste phpMyAdmin unter *http://localhost/phpmyadmin/* zur Verfügung stehen, andernfalls müssen Sie die URL gemäß Ihrer eigenen Konfiguration eingeben. Je nachdem, wie Sie die Benutzeranmeldung eingerichtet haben, müssen Sie nach Aufruf der URL Benutzername und Kennwort eingeben oder auch nicht. Anschließend gelangen Sie auf die in Abbildung 4-3 gezeigte phpMyAdmin-Startseite.

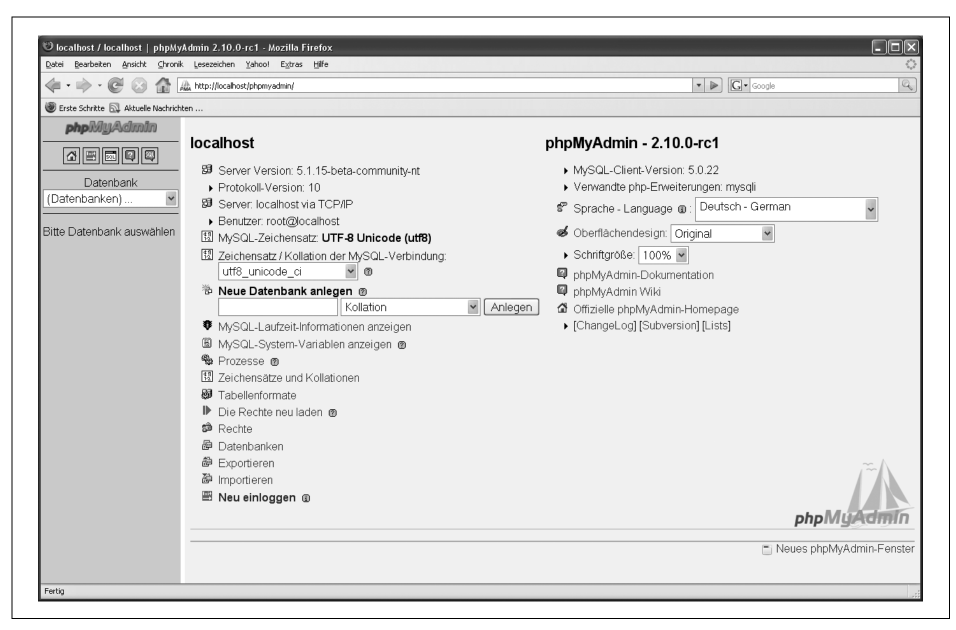


Abbildung 4-3: Die Startseite von phpMyAdmin

Der Bildschirm ist in zwei Frames unterteilt: Im schmalen linken Frame können Sie eine Datenbank aus dem Pull-down-Menü auswählen. Rechts finden Sie auf der Startseite die beiden Rubriken *MySQL* mit grundlegenden Operationen und Informationen zum Datenbankserver sowie phpMyAdmin mit einigen Grundeinstellungen für den phpMyAdmin-Client selbst. Tabelle 4-3 zeigt zunächst eine Übersicht über die einzelnen Punkte der Spalte *MySQL*. Falls verfügbar, wird zusätzlich der entsprechende Befehl aufgelistet, den Sie für dasselbe Ergebnis im mysql-Kommandozeilenclient eingeben müssten. Einige dieser Optionen werden in Kapitel 9, *MySQL-Administration*, genauer behandelt.

Tabelle 4-3: »MySQL«-Optionen auf der phpMyAdmin-Startseite

Funktion	Kommandozeilenvariante	Beschreibung
Zeichensatz/Kollation der MySQL-Verbindung	`SET NAMES ...`	Stellt den Zeichensatz ein, den der Client zur Kommunikation mit dem MySQL-Server verwendet. Sollte mit dem (meist durch die Website selbst) im Browser eingestellten Zeichensatz übereinstimmen.
Neue Datenbank anlegen	`CREATE DATABASE ...`	Eine neue Datenbank mit der angegebenen Kollation (Sprach-Sortierreihenfolge) erstellen.
MySQL-Laufzeit-Informationen anzeigen	`SHOW STATUS`	Ausführliche Variante der Statusanzeige.
MySQL-System-Variablen anzeigen	`SHOW VARIABLES`	In Systemvariablen gespeicherte MySQL-Einstellungen und -Eigenschaften anzeigen.
Prozesse	`SHOW PROCESSLIST`	Alle aktiven Verbindungen des MySQL-Servers anzeigen.
Zeichensätze und Kollationen	`SHOW CHARSET/SHOW COLLATION`	Übersicht aller Zeichensätze und Sprach-Sortierreihenfolgen (Kollationen).
Tabellenformate	`SHOW ENGINES`	Liste aller vom aktuellen MySQL-Server unterstützten Storage Engines oder Tabellentypen (siehe nächstes Kapitel).
Die Rechte neu laden	`FLUSH PRIVILEGES`	Benutzer und ihre Berechtigungen nach einer Änderung neu laden.

Rechte	`SHOW PRIVILEGES`
	`CREATE USER ...`
	`GRANT ...      usw.`
		Komfortable Oberfläche zur Anzeige und Änderung der MySQL-Benutzerrechte.

Datenbanken	`SHOW DATABASES`	Liste aller Datenbanken des MySQL-Servers.
Exportieren	Dienstprogramm *mysqldump* u.a.	Datenbanken in verschiedenen Formaten exportieren (siehe Kapitel 9).

Importieren	`source`
	`mysql ... <Quelldatei`
	`SELECT ... FROM INFILE`
	`usw.`
		Tabellenstrukturen und -daten aus SQL-Dateien oder formatierten Textdateien importieren (siehe Kapitel 9).

Neu einloggen[a]	`CONNECT`	Neue Verbindung zum MySQL-Server herstellen.

[a] Der Internet Explorer hat einen Bug, der die erneute Anmeldung desselben MySQL-Users verhindert.

In der Spalte *phpMyAdmin* können Sie dagegen Einstellungen für den Client selbst vornehmen und einige Informationen erhalten:

- Unter *Sprache - Language* wird die Sprache von phpMyAdmin selbst eingestellt – für alle Beispiele in diesem Buch wurde natürlich *Deutsch - German* gewählt, aber wie Sie sehen, unterstützt das Tool noch zahlreiche weitere Sprachen.
- *MySQL-Zeichensatz* zeigt den (hier nicht änderbaren) Zeichensatz an, den der Server selbst standardmäßig verwendet.

- *Zeichensatz/Kollation der MySQL-Verbindung* ermöglicht Ihnen die Auswahl, welchen Zeichensatz und welche Kollation phpMyAdmin bei der Kommunikation mit dem Server verwenden soll.
- Unter *Oberflächendesign* können Sie ein Farbschema für phpMyAdmin auswählen; mitgeliefert werden die beiden Einstellungen *Original* und *Darkblue/orange*. Die Auswahl hat nicht nur ästhetische Bedeutung, sondern ermöglicht je nach Umgebungslicht und Monitortyp eventuell eine bessere Kontrastwirkung. Im Internet sind weitere Themen erhältlich.
- Die *Schriftgröße* lässt sich prozentual verkleinern oder vergrößern.
- Die über den gleichnamigen Link erreichbare lokale *phpMyAdmin-Dokumentation* wird automatisch mit phpMyAdmin geliefert.
- *phpMyAdmin Wiki* öffnet ein neues Browserfenster, in dem das vor Kurzem gegründete Wiki der phpMyAdmin-Entwickler angezeigt wird. Das Wiki-Konzept kennen Sie wahrscheinlich von der Online-Enzyklopädie Wikipedia: Die Inhalte können direkt online von jedem beliebigen Besucher geändert oder ergänzt werden.
- Die restlichen Links verweisen auf die Websites des phpMyAdmin-Projekts selbst: Neben der offiziellen *Homepage* sind auch das *ChangeLog* (Liste der Änderungen pro Version), das *Subversion*-Repository (Quellcode-Archiv mit Versionsverwaltung) sowie *Lists* (die phpMyAdmin-Mailinglisten) zu erreichen.

Unter den beiden Spalten werden eventuell Konfigurations- oder Sicherheitsprobleme von phpMyAdmin oder der zugrunde liegenden PHP-Installation angezeigt. Besonders kritische Fehler (zum Beispiel Zugriff mit einem leeren Passwort) werden in Rot dargestellt und sollten unverzüglich behoben werden.

Sobald Sie links eine Datenbank auswählen, werden die Namen aller ihrer Tabellen als Hyperlinks angezeigt. Auch im rechten Frame erscheint eine Übersicht über sämtliche Tabellen. In Abbildung 4-4 wird dies für die Datenbank *gewinnspiel* aus dem vorigen Kapitel dargestellt. In der Spalte *Aktion* finden Sie Schaltflächen für den schnellen Zugriff auf sechs wichtige Tabellenoperationen: *Anzeigen* stellt die Inhalte der Tabelle dar, *Struktur* ermöglicht Änderungen an der Tabellenstruktur selbst (Namen, Datentypen, Reihenfolge der Spalten usw.), mittels *Suche* können Sie die Tabelle nach diversen Kriterien durchsuchen, *Einfügen* dient dem Hinzufügen neuer Datensätze, *Leeren* löscht sämtliche Inhalte der Tabelle und *Löschen* sogar die Tabelle selbst. Für die beiden letzteren Operationen wird allerdings zuvor eine JavaScript-basierte Sicherheitsabfrage durchgeführt.

Unter der Tabellenübersicht finden Sie ein Formular zum Erstellen einer neuen Tabelle innerhalb der aktuellen Datenbank. Dazu müssen Sie einen Namen und die gewünschte Anzahl der Felder pro Datensatz eingeben. Sobald Sie auf *OK* klicken, gelangen Sie auf eine neue Seite, die weiter unten beschrieben wird.

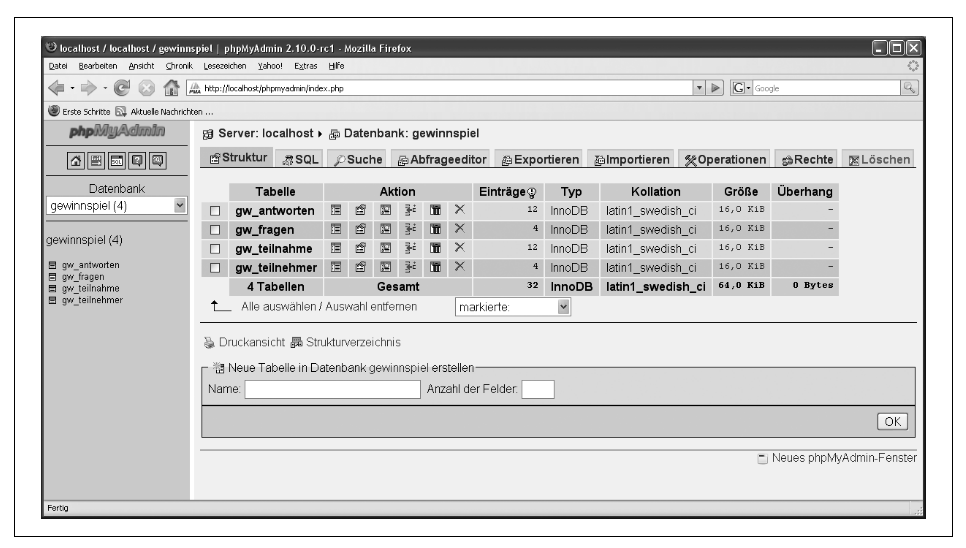


Abbildung 4-4: Übersichtsseite einer Datenbank in phpMyAdmin



In vielen Browsern (zum Beispiel in Firefox, der für die Beispiele in diesem Buch verwendet wird, oder auch im Microsoft Internet Explorer) werden die Funktionen aller phpMyAdmin-Schaltflächen als Tooltipps angezeigt, sobald Sie mit der Maus darüberfahren. Sämtliche Schaltflächenbezeichnungen im vorliegenden Buch beziehen sich auf diese Tooltipps.

Tabellenoptionen

Wenn Sie im linken Frame eine Tabelle auswählen, sieht der Hauptframe so aus, wie es in Abbildung 4-5 für die Tabelle *gw_fragen* gezeigt wird. Die Seite entspricht der Auswahl *Struktur* auf der Hauptseite einer Datenbank.

Ganz oben auf der Seite befinden sich einige Links auf wichtige Datenbankbereiche und Tabellenoptionen. In der ersten Zeile werden Links auf die aktuelle Hierarchie aus Host, Datenbank und Tabelle angezeigt. Die zweite Zeile enthält Links auf diverse Seiten zur Tabellenbearbeitung:

- *Anzeigen*: Stellt die Inhalte der Tabelle mit Blätter- und Sortiermöglichkeiten dar, standardmäßig je 30 Datensätze pro Seite.
- *Struktur* (aktuelle Seite): Stellt den Grundaufbau, aber nicht den Inhalt der Tabelle dar.
- *SQL*: Ermöglicht die manuelle Eingabe oder den Datei-Upload von SQL-Abfragen. Auf der *Struktur*-Seite befindet sich ebenfalls ein solches Formular, das in Kürze beschrieben wird.

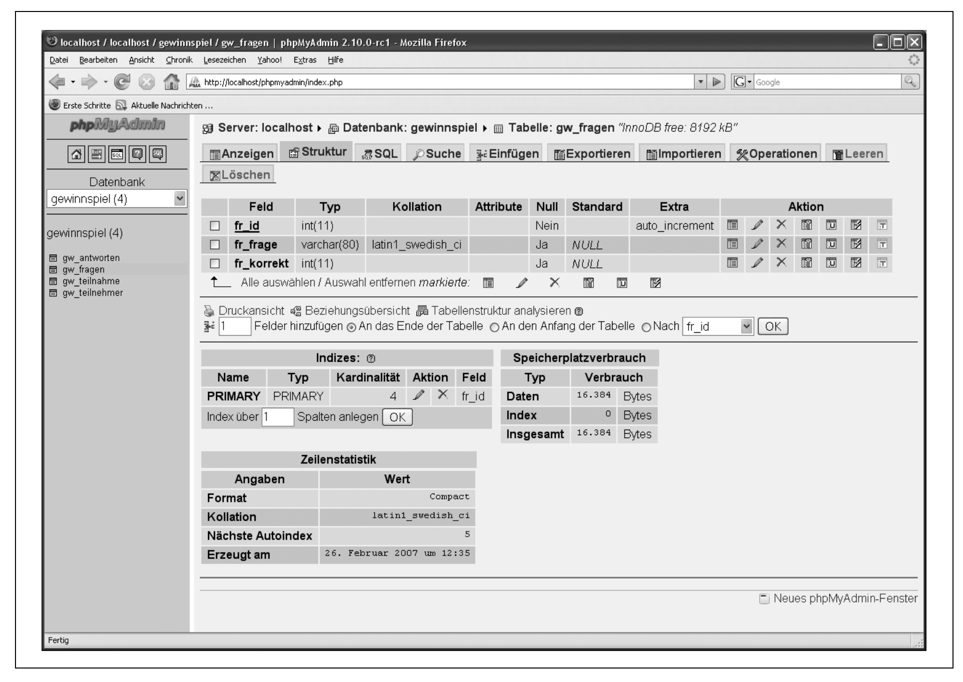


Abbildung 4-5: Startseite einer Datenbanktabelle in phpMyAdmin

- *Suche*: Ein Formular mit verschiedenen Optionen zur Suche in der Datenbank.
- *Einfügen*: Neue Datensätze in die aktuelle Tabelle eingeben.
- *Exportieren*: Exportiert die Struktur und/oder die Daten der Tabelle in verschiedenen Formaten mit diversen Optionen.
- *Importieren*: Importiert Daten aus SQL-Dateien oder formatierten Textdateien.
- *Operationen*: Änderung von Tabelleneigenschaften wie Name, Typ oder Kollation.
- *Leeren*: Entfernen aller Inhalte der Tabelle (mit Sicherheitsabfrage).
- *Löschen*: Entfernen der gesamten Tabelle und ihrer Inhalte, ebenfalls mit Nachfrage zur Absicherung.

Einige der genannten Optionen werden im Folgenden näher erläutert, andere erst in späteren Kapiteln, zu deren Thematik sie besser passen.

Tabellenstruktur

Wie bereits erwähnt, stellt die erste Tabellenbearbeitungsseite die Struktur einer Tabelle dar. Sie erscheint, wenn Sie im linken Frame eine Tabelle auswählen oder wenn Sie den Link *Struktur* auf einer anderen Tabellenseite anklicken. In Abbildung 4-6 sehen Sie die Strukturansicht der Tabelle *gw_antworten*.

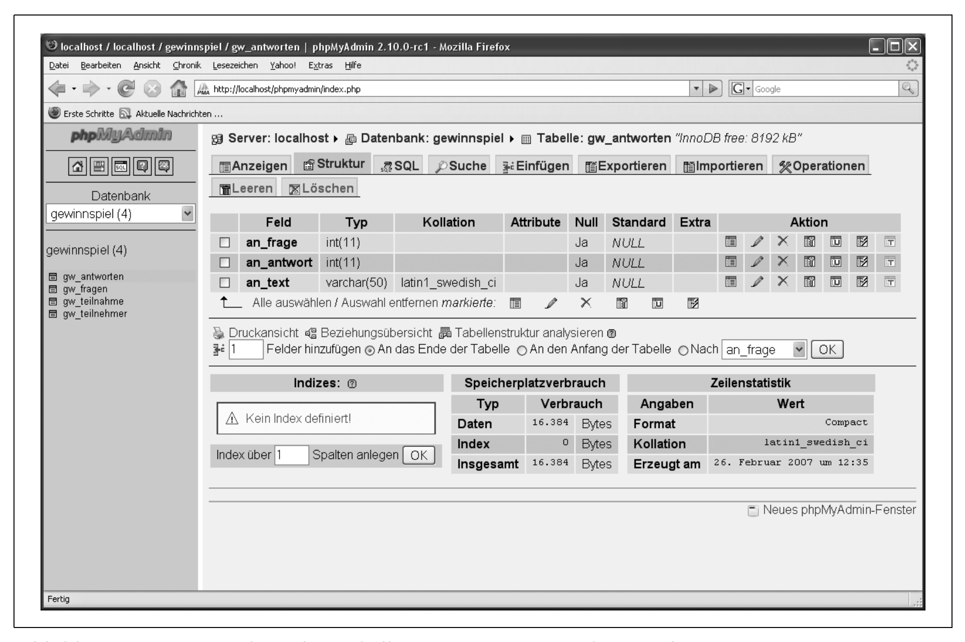


Abbildung 4-6: Die Struktur der Tabelle gw_antworten in phpMyAdmin

Oben werden zunächst die wichtigsten Eigenschaften aller Tabellenspalten angezeigt:

- *Feld*: Name der Spalte.
- *Typ*: SQL-Datentyp der Spalte.
- *Kollation*: Sprach-Sortierreihenfolge von Spalten mit Textdatentypen.
- *Attribute*: Zusatzeigenschaften einer Spalte, zum Beispiel `UNSIGNED` (vorzeichenlos).
- *Null*: Gibt an, ob Felder in der Spalte leer bleiben dürfen oder nicht.
- *Standard*: Standardwert für die Felder der Spalte, falls definiert.
- *Extra*: Falls die Spalte automatisch hochgezählt wird, steht hier `auto_increment`.
- *Aktion*: Schaltflächen für Operationen mit dieser Spalte. Von links nach rechts stehen folgende Optionen zur Verfügung: *Zeige nur unterschiedliche Werte* – jeder in der Spalte vorkommende Wert wird einmal angezeigt, *Ändern* – Spalteneigenschaften modifizieren, *Löschen* – Spalte und ihre Inhalte entfernen, *Primärschlüssel* – diese Spalte als Primärschlüssel der Tabelle einstellen, *Unique* – festlegen, dass jedes Feld der Spalte einen eindeutigen Wert enthalten muss, *Index* – einen Index für die Spalte einrichten, *Volltext* – einen Volltextindex erstellen (nur für einfache Texttypen in *MyISAM*-Tabellen).



Die einzelnen Optionen dieser Übersicht werden weiter unten genauer erläutert, wenn es darum geht, sie manuell einzugeben beziehungsweise festzulegen.

Unter der Tabellenansicht finden Sie einen Satz Schaltflächen, die den soeben beschriebenen Aktionen entsprechen. Auf diese Weise lassen sich die Aktionen auf mehrere per Kontrollkästchen ausgewählte Spalten gleichzeitig anwenden. In diesem Zusammenhang sind auch die beiden Links *Alle auswählen* und *Auswahl entfernen* sehr praktisch.

Im nächsten Abschnitt finden Sie zunächst drei Links: *Druckansicht* stellt die Tabellenstruktur auf einer nüchternen, druckoptimierten Seite ohne Einstellungsmöglichkeiten dar und enthält zusätzlich die passende Schaltfläche zum Drucken. Die *Beziehungsübersicht* stellt definierte Verknüpfungen zur referenziellen Integrität dar – Näheres darüber erfahren Sie in Kapitel 5. Der nächste Punkt, *Tabellenstruktur analysieren*, vergleicht die Datentypen der Tabelle mit den tatsächlich enthaltenen Feldwerten und macht gegebenenfalls (nicht immer praxistaugliche!) Verbesserungsvorschläge.

Als Nächstes erhalten Sie die Möglichkeit, eine frei wählbare Anzahl neuer Spalten (*Felder*) der Tabelle hinzuzufügen. Sie können sich aussuchen, ob sie am Anfang der Tabelle, an ihrem Ende oder hinter einer bestimmten vorhandenen Spalte eingefügt werden sollen. Wenn Sie die gewünschten Einstellungen vorgenommen und die Schaltfläche *OK* angeklickt haben, gelangen Sie auf eine neue Seite, die der weiter unten beschriebenen Eingabeseite für neue Tabellen entspricht.

Der nachfolgende Abschnitt gibt Auskunft über die in der Tabelle definierten *Indizes*. Ausführliche Informationen über Schlüssel und Indizes erhalten Sie im nächsten Kapitel.

Tabelleninhalt

Wenn Sie eine Tabelle ausgewählt haben und den Link *Anzeigen* anklicken, werden die Inhalte der Tabelle angezeigt; in Abbildung 4-7 sehen Sie die Tabelle *gw_fragen* als Beispiel. Oben erscheint der Wortlaut der SQL-Abfrage, die zum aktuellen Ausschnitt der Tabellendaten geführt hat. Per Voreinstellung werden jeweils 30 Datensätze pro Seite angezeigt, was durch eine `LIMIT`-Klausel erreicht wird.

Oberhalb der eigentlichen Daten (und ein weiteres Mal darunter) können Sie festlegen, welche und wie viele Datensätze angezeigt werden sollen; ein Klick auf die Schaltfläche *Zeige* bestätigt Ihre Auswahl. Besitzt die Tabelle mehr Zeilen, als bei der gewählten Anzahl auf die aktuelle Seite passen, werden zusätzliche Schaltflächen zum Blättern bereitgestellt: < und > blättern je eine Seite zurück beziehungsweise vor, während << und >> an den Anfang beziehungsweise das Ende der derzeitigen Sortierreihenfolge springen.

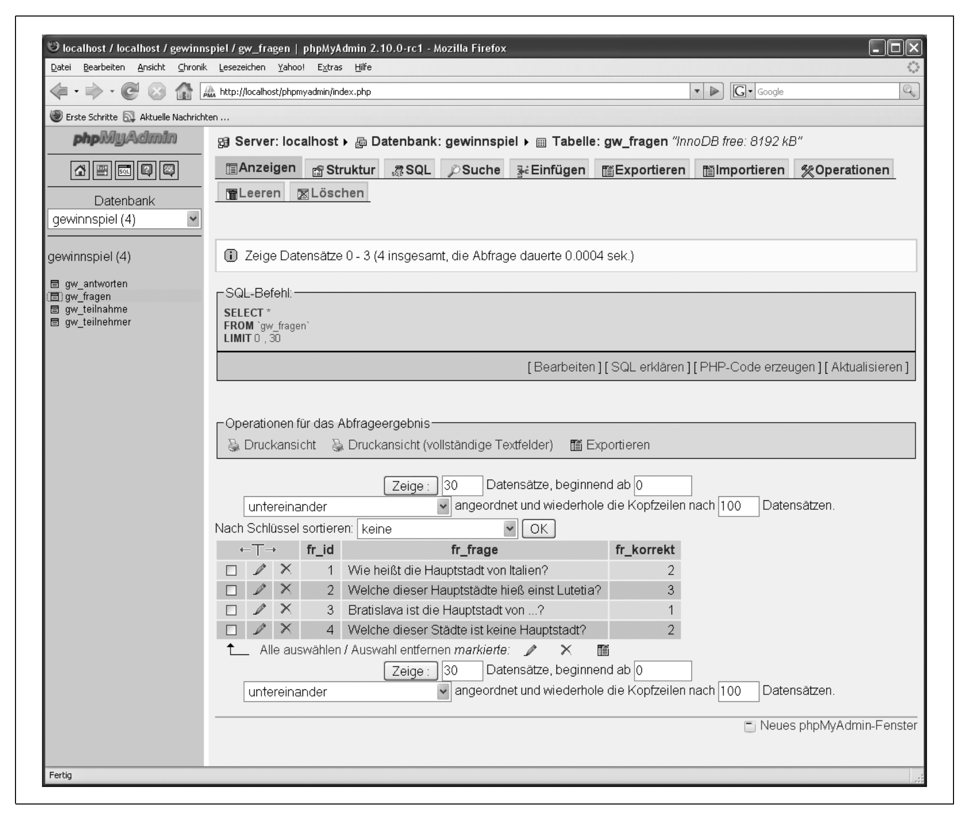


Abbildung 4-7: Darstellung des Tabelleninhalts von gw_fragen in phpMyAdmin

In der nächsten Zeile können Sie sich weitere Darstellungsoptionen aussuchen: Der Standardmodus *untereinander* stellt die Datensätze in vertikaler und die Spalten in horizontaler Ansicht dar. Die Option *horizontal (gedrehte Bezeichner)* dreht die Spaltenbezeichnungen um 90°, was bei Spalten mit schmalen Inhalten einiges an Platz spart – allerdings funktioniert das bisher nur im Internet Explorer. Der Modus *nebeneinander* schließlich vertauscht die Tabellenachsen – die Datensätze stehen in Spalten, ihre einzelnen Felder untereinander.

Als Nächstes geht es um die Sortierreihenfolge der angezeigten Datensätze: Das Pull-down-Menü *Nach Schlüssel sortieren* ermöglicht die Auswahl des Primärschlüssels und anderer indizierter Spalten als Sortierkriterium. Viel intuitiver und einfacher ist es dagegen, den Titel der Spalte anzuklicken, nach der sortiert werden soll. Sobald Sie dieselbe Spalte zum zweiten Mal anklicken, wird übrigens die Sortierrichtung umgekehrt; ein kleines Dreieck neben dem Spaltentitel markiert die aktuelle Einstellung.



Oft ist es wünschenswert, mehrere Datensätze, bei denen das aktuelle Sortierkriterium denselben Wert hat, in sich nach einem zweiten Kriterium zu sortieren. Ein häufiger Fall ist etwa das Sortieren nach Vornamen bei identischem Nachnamen. In phpMyAdmin funktioniert dies leider nur per manueller Eingabe einer SQL-Abfrage. Näheres erfahren Sie in Kapitel 6.

Für jede Zeile wird ein Kontrollkästchen zum Auswählen angezeigt, damit Sie mithilfe der Schaltflächen unter der Tabelle mehrere Datensätze *bearbeiten* (Stiftsymbol), *löschen* (rotes X) beziehungsweise *exportieren* (Tabellensymbol) können. Die Link-Symbole zum *Ändern* sowie zum *Löschen* finden Sie zusätzlich bei jedem einzelnen Datensatz.

Manuelle SQL-Eingabe

Auf der Registerkarte *SQL* (siehe Abbildung 4-8) finden Sie ein umfangreiches Textfeld zur freien Eingabe von SQL-Abfragen. Rechts daneben befindet sich ein Auswahlfeld mit sämtlichen Spalten der aktuellen Tabelle. Wenn Sie darin einen Feldnamen anklicken und anschließend die daneben liegende Pfeilschaltfläche betätigen (Kurzfassung: Doppelklick), wird dieser Name – ordentlich in 'Backticks' eingefasst – an die Cursorposition des Eingabefelds gesetzt. Der automatisch vorgegebene Eintrag dient der Auswahl sämtlicher Spalten (*) und aller Datensätze (Kriterium `WHERE 1`); er entspricht der Tabellenoption *Anzeigen*.

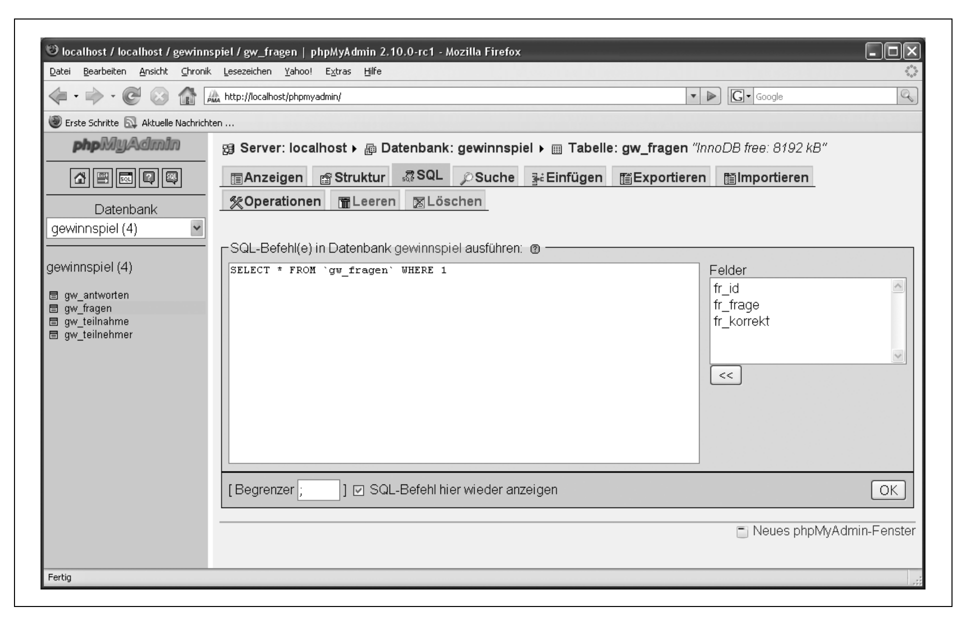


Abbildung 4-8: Der phpMyAdmin-Bereich zur manuellen Eingabe von SQL-Abfragen

Sie können diese Registerkarte vor allem nutzen, um die in Kapitel 6 vorgestellten SQL-Auswahlabfragen auf komfortable Weise nachzuvollziehen und ihre Ergebnisse in übersichtlicher Form zu lesen.

Tabellenoperationen

Wenn Sie den Link *Operationen* am oberen Rand von Tabellenseiten anklicken, können Sie einige interessante Änderungen an Tabellen durchführen. Abbildung 4-9 zeigt alle Optionen im Überblick.

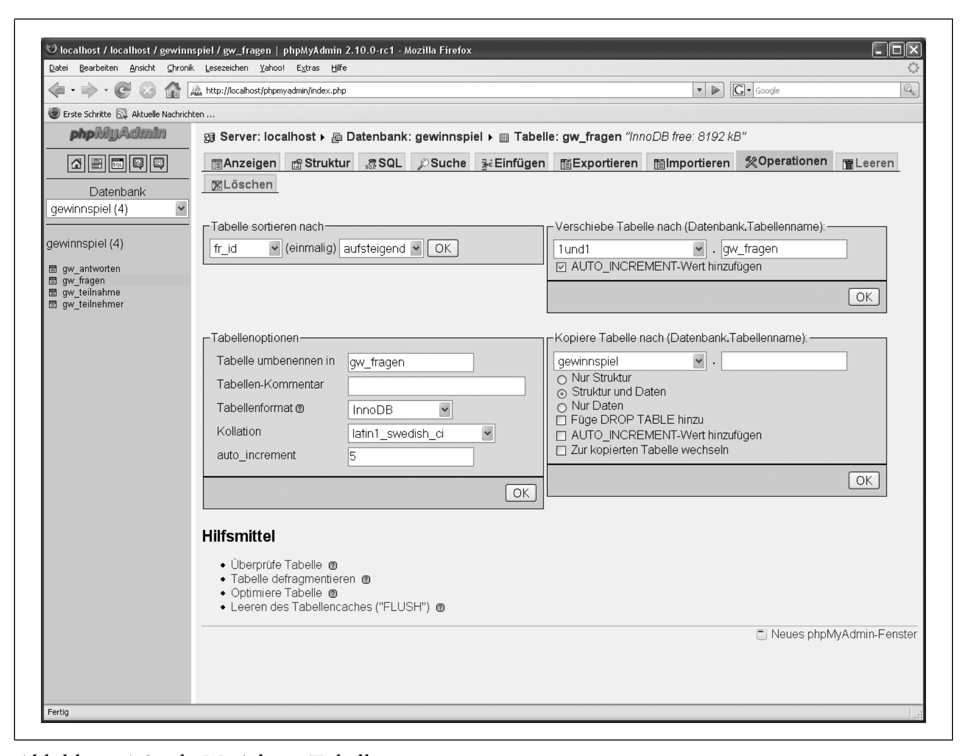


Abbildung 4-9: phpMyAdmin-Tabellenoperationen

Tabelle sortieren nach weist der Tabelle eine automatische Vorsortierung nach dem gewählten Kriterium zu. Das kann nützlich sein, um die Ausgabe der Tabelle in der gewünschten Standardsortierung zu beschleunigen.

Mithilfe der Option *Tabelle umbenennen in* können Sie der Tabelle einen neuen Namen zuweisen. Als Nächstes können Sie den *Tabellen-Kommentar* – eine beliebige Beschreibung – ändern. Darunter lässt sich das *Tabellenformat* nachträglich modifizieren. Näheres über die verschiedenen MySQL-Tabellentypen erfahren Sie im nächsten Kapitel. Anschließend kann die voreingestellte *Kollation* der Tabelle geändert werden; Spalten, die eine eigene Einstellung besitzen, sind davon nicht

betroffen. Zuletzt können Sie in diesem Bereich bestimmen, welcher Wert für den nächsten *auto_increment*-Eintrag eingefügt werden soll.

Verschiebe Tabelle nach geht über das Umbenennen innerhalb derselben Datenbank hinaus und ermöglicht auch den Umzug der Tabelle in eine andere Datenbank auf demselben MySQL-Server.

Die Option *Kopiere Tabelle nach* besitzt mehr Einstellungsmöglichkeiten als die bisher genannten Punkte: Unter der Angabe von Zieldatenbank und -tabelle können Sie bestimmen, ob *Nur Struktur*, *Struktur und Daten* oder *Nur Daten* kopiert werden sollen. Letzteres erfordert als Ziel eine Tabelle mit identischen Spaltendefinitionen. Die Zusatzoption *Füge DROP TABLE hinzu* löscht eine eventuell vorhandene Tabelle gleichen Namens. *AUTO_INCREMENT-Wert hinzufügen* erstellt einen nummerierten Schlüssel für die neuen Daten. *Zur kopierten Tabelle wechseln* zeigt nach getaner Arbeit die soeben erstellte Kopie an.



Für die gesamte Datenbank gibt es eine vergleichbare Seite. Sie erreichen sie, wenn Sie den Link *Operationen* auf der Übersichtsseite einer Datenbank anklicken. Dort können Sie eine neue Tabelle erstellen und die Datenbank umbenennen oder kopieren. Außerdem lässt sich die Kollation der Datenbank selbst ändern, was nur Tabellen betrifft, für die keine explizite Kollation definiert wurde.

Datenbanken und Tabellen erstellen

Wechseln Sie auf die Startseite (Haussymbol im linken Frame), wenn Sie eine neue Datenbank anlegen möchten. Sie müssen einen Namen für die Datenbank angeben; natürlich darf noch keine Datenbank mit dem gewünschten Namen existieren. Zusätzlich können Sie eine *Kollation*, also eine Standard-Sortierreihenfolge, auswählen. Sobald Sie auf *OK* klicken, erscheint die Übersichtsseite der neuen Datenbank – natürlich noch ohne Tabellen.

Als Beispiel soll eine separate Datenbank für eine Promotion-Aktion erstellt werden: eine Liste kostenloser Zusatzleistungen, die ab einem angegebenen Buchungspreis gewährt werden. Geben Sie den Datenbanknamen »promo« ein. Für die Sprache Deutsch gibt es zwei verschiedene Kollationen: *latin1_german1_ci* sortiert gemäß Wörterbuch, so dass die Umlaute *ä*, *ö* und *ü* mit den Buchstaben *a*, *o* und *u* gleichgesetzt werden. *latin1_german2_ci* sortiert dagegen nach der Telefonbuch-Reihenfolge, die die Umlaute als *ae*, *oe* und *ue* behandelt. Im vorliegenden Fall sollten Sie *latin1_german1_ci* wählen.

Im nächsten Schritt können Sie die erste Tabelle in der neuen Datenbank erstellen. Geben Sie dazu unter *Neue Tabelle in der Datenbank promo erstellen* einen Tabellennamen (hier »pr_angebote«) und die Anzahl der Felder beziehungsweise Spalten (3) ein. Nach einem Klick auf *OK* wird eine Seite zur Definition der Tabellenspalten angezeigt (siehe Abbildung 4-10).

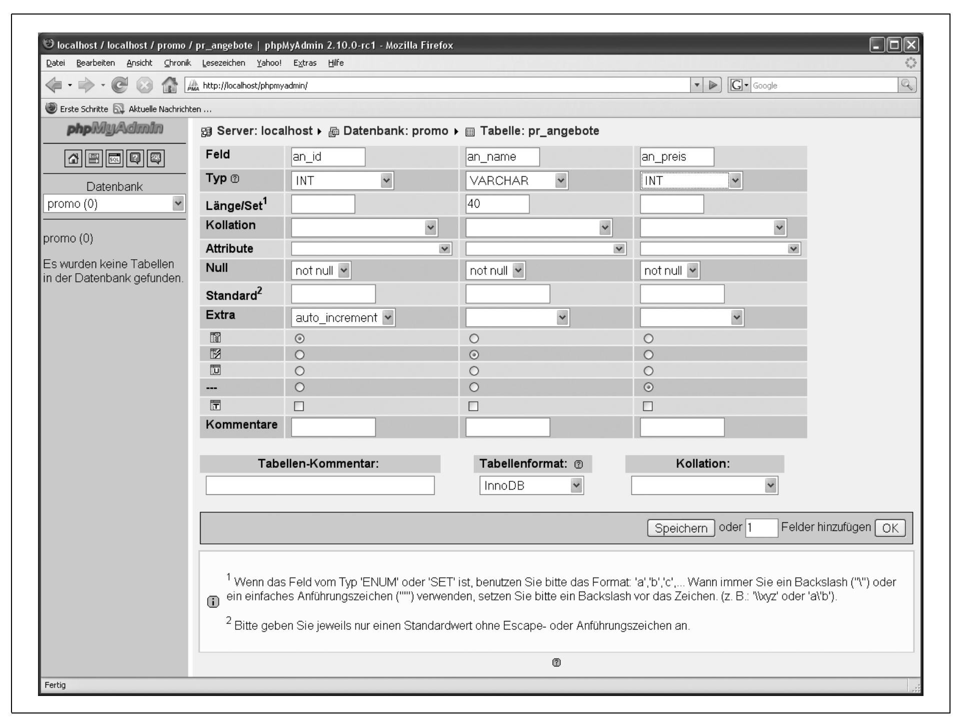


Abbildung 4-10: Definition der Felder einer neuen Tabelle in phpMyAdmin

Das Eingabeformular enthält für jedes Feld die folgenden Eingabemöglichkeiten:

- *Feld*: Name der Spalte.
- *Typ*: Datentyp der Spalte – eine genaue Beschreibung aller SQL-Datentypen folgt im nächsten Kapitel.
- *Länge/Set*: Byte- oder Zeichenanzahl des Datentyps beziehungsweise Werteliste von Aufzählungstypen.
- *Kollation*: Sortierreihenfolge gemäß Zeichensatz und Sprache, nur für Textdatentypen – wenn Sie hier nichts auswählen, wird automatisch die Kollation der gesamten Tabelle verwendet.
- *Attribute* – Auswahlmöglichkeiten für Zahlentypen: *UNSIGNED* (vorzeichenlos) und *UNSIGNED ZEROFILL* (vorzeichenlos, führende Stellen bis zur Gesamtbreite mit Nullen auffüllen).
- *Null*: Gibt an, ob das Feld leer bleiben darf (*null*) oder nicht (*not null*, Standard).
- *Standard*: Automatischer Vorgabewert für Felder der Spalte.
- *Extra*: Auswahl des Attributs *auto_increment* für automatisch durchnummerierte (Schlüssel-)Felder.

- *Indizes* zur Auswahl: *Primärschlüssel*, normaler *Index*, *unique* (jedes Feld der Spalte muss einen einmaligen Wert besitzen), kein Index (Standard), *Volltext* (nur in MyISAM-Tabellen möglich).
- *Kommentare*: Eine optionale Beschreibung jeder Spalte zum internen Gebrauch.

Füllen Sie das Formular gemäß den Angaben in Tabelle 4-4 aus.

Tabelle 4-4: Spalten der neuen Tabelle pr_angebote

Spaltenname	Datentyp	Weitere Optionen
an_id	INT	auto_increment, Primärschlüssel
an_name	VARCHAR	Länge 40, Index
an_preis	INT	–

Das *Tabellenformat* können Sie auf *Standard* stehen lassen (ergibt unter Unix *MyISAM* und unter Windows *InnoDB*). Wenn Sie möchten, können Sie aber auch explizit *MyISAM* wählen, da für diese Tabelle keine Transaktionen vorgesehen sind. Zusätzlich können Sie einen beliebigen *Kommentar* für die Tabelle eingeben sowie eine *Kollation* auswählen. Letzteres ist hier nicht erforderlich, da die Kollation der Datenbank automatisch für alle Tabellen gilt, die keine eigene Einstellung besitzen. Klicken Sie zum Schluss auf *Speichern*, um die Einstellungen zu akzeptieren. Es erscheint die Strukturseite der Tabelle; zusätzlich wird die soeben ausgeführte SQL-Abfrage zur Tabellenerstellung angezeigt. Sie sieht wie folgt aus (im nächsten Kapitel werden Tabellenerstellungsabfragen erläutert):

```
CREATE TABLE `pr_angebote` (
  `an_id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  `an_name` VARCHAR( 40 ) NOT NULL ,
  `an_preis` INT NOT NULL ,
  INDEX ( `an_name` )
) ENGINE = innodb;
```

Der letzte Schritt besteht natürlich darin, Datensätze in die Tabelle einzufügen. Die klassische Methode ist die manuelle Eingabe. Klicken Sie dazu auf den Link *Einfügen* am oberen Fensterrand. Es erscheint eine Eingabeseite wie in Abbildung 4-11.

Hier können Sie jeweils einen Datensatz eingeben, nach Entfernen der Auswahl *Ignorieren* (geschieht automatisch, sobald Sie im unteren Bereich etwas eingeben) auch zwei. Wenn die Eingabe beendet ist, können Sie sich aussuchen, ob Sie mittels *zurück* zur vorigen Ansicht wechseln oder *anschließend einen weiteren Datensatz einfügen* möchten. Geben Sie die in Tabelle 4-5 gezeigten Datensätze ein, indem Sie vor dem Klick auf *OK* wählen, dass Sie einen weiteren Datensatz eingeben möchten. Nach der ersten Auswahl dieser Option sollte sie automatisch bestehen bleiben. Das nicht angegebene Feld *an_id* können (und sollten!) Sie jeweils leer lassen, da es per *auto_increment* automatisch ausgefüllt wird.

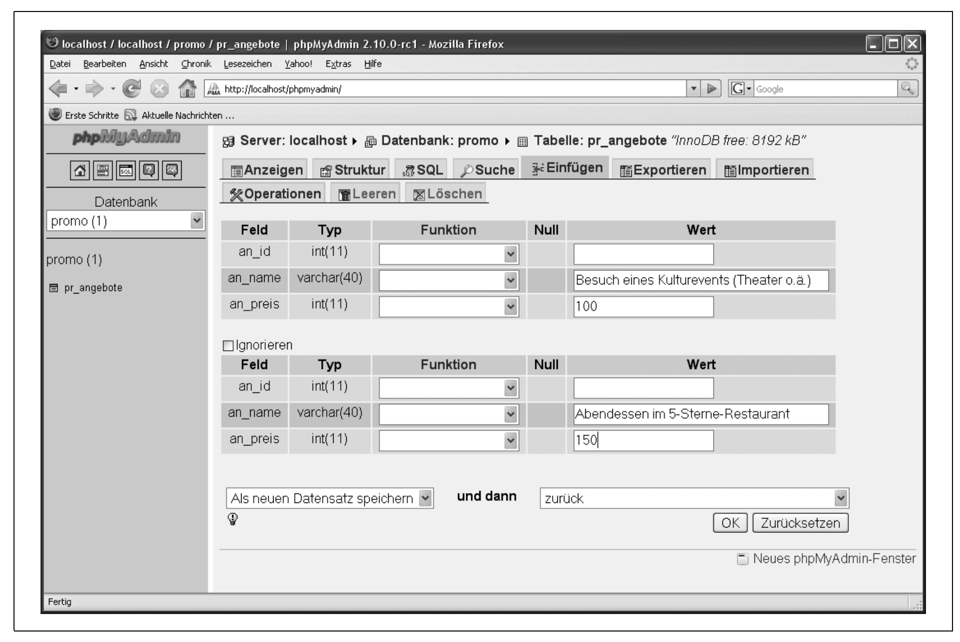


Abbildung 4-11: Eingabe neuer Datensätze in phpMyAdmin

Tabelle 4-5: Datensätze für die Tabelle pr_angebote

an_name	an_preis
Besuch eines Kulturevents (Theater o.Ä.)	100
Abendessen im 5-Sterne-Restaurant	150
Kostenlose Stadtrundfahrt	200
ÖPNV-Ticket für Aufenthaltsdauer	250
Kostenloser Mietwagen (2 Tage)	300



Die angesprochenen Funktionen zur nachträglichen Änderung von Tabellenstrukturen und -inhalten sind übrigens weitgehend mit den hier vorgestellten Neueingabeseiten identisch – natürlich mit dem einen Unterschied, dass die Eingabefelder bereits Werte enthalten, die Sie ändern können.

In diesem Kapitel:

- Der Datenbankentwurf
- Datenbanken und Tabellen erstellen
- MySQL-Datentypen
- Schlüssel und Indizes
- Daten einfügen

KAPITEL 5

Datenbanken entwerfen und erstellen

Wo Inhalt ist, fügen sich die Formen von selbst.
Leo Tolstoi

In diesem Kapitel erfahren Sie, wie MySQL-Datenbanken und -Tabellen konzipiert und erstellt werden. Im ersten Abschnitt geht es um die Planung und den Entwurf von Datenbanken, anschließend wird die praktische Erzeugung von Datenbanken und Tabellen mithilfe von SQL-Abfragen vorgestellt.



Alle SQL-Abfragen in diesem und den nächsten beiden Kapiteln werden anhand des Kommandozeilenclients mysql vorgestellt. Daher enden sie alle mit einem Semikolon, das in anderen Umgebungen wie etwa phpMyAdmin nicht nötig ist – auch dann, wenn die Eingabeaufforderung `mysql>` nicht abgebildet wird.

Der Datenbankentwurf

Bevor Sie damit beginnen, SQL-Abfragen zur Erzeugung von Datenbanktabellen einzugeben, sollten Sie sich die Zeit nehmen, ihren Aufbau sorgfältig zu planen und zu entwerfen. Es ist überaus ärgerlich, wenn man erst nach dem Einfügen von einigen Hundert Datensätzen bemerkt, dass das Datenbankdesign nicht allen Anforderungen gerecht wird. Deshalb werden hier zwei bekannte Ansätze für den Entwurf relationaler Datenbanken vorgestellt: das klassische Entity-Relationship-Modell sowie die sogenannten Normalisierungsregeln.

Intuitiver Ansatz

Der erste Schritt zur Erstellung von Tabellen besteht darin, sich zu fragen – und am besten auf einem großen, leeren Blatt zu notieren –, welche Informationen überhaupt in der Datenbank gespeichert werden sollen.

Um diesen Abschnitt nicht zu überfrachten, wird hier nur ein Teilaspekt der künftigen Beispieldatenbank betrachtet: die Speicherung der verschiedenen Hotels. Über jedes einzelne Hotel sollen folgende Informationen gespeichert werden (konzeptionelle Einschränkungen des Beispiels wurden bereits im Vorwort erörtert):

- eindeutige Nummer (automatisch durchnummeriert) als Primärschlüssel
- Name des Hotels
- Stadt, in der es sich befindet
- Preis eines Einzelzimmers pro Nacht
- Preis eines Doppelzimmers pro Nacht
- Bad: ohne, nur WC, Dusche oder Bad
- Verpflegung: ohne, Frühstück, Halbpension oder Vollpension
- Postanschrift des Hotels
- URL der Hotel-Website

Die Spalten *Stadt* und *Postanschrift* sind übrigens keineswegs redundant: Die Stadt muss separat gespeichert werden, weil sie das für die Buchung relevante Reiseziel darstellt, während die Postanschrift lediglich eine hilfreiche Kundeninformation ist. Um Flug- und Hotelangebote miteinander in Einklang zu bringen, ist es im Übrigen sinnvoll, nicht den Namen der Stadt zu speichern, sondern einen Bezug auf eine Städte-Tabelle. Es zeigt sich also, dass für die Hotelinformationen der Zugriff auf eine weitere Tabelle erforderlich ist. Der genaue Aufbau dieser fremden Tabelle tut hier noch nichts zur Sache; sie enthält zusätzliche Touristeninformationen über die einzelnen Städte und wird weiter unten vorgestellt. Wichtig ist hier nur, dass sie ein automatisch durchnummeriertes Feld als Primärschlüssel enthält; der jeweilige Wert dieses Felds bildet den Eintrag in der *Stadt*-Spalte der Hotel-Tabelle.

Der nächste Schritt besteht darin, sich zu überlegen, welchen Datentyp die einzelnen Spalten besitzen sollen. Genaueres über die MySQL-Felddatentypen erfahren Sie weiter unten; hier geht es zunächst um eine allgemeine, eher intuitive Entscheidung. Sie basiert auf der Frage, welche Arten von Daten in den einzelnen Feldern gespeichert werden sollen. Diese Überlegung stellt sich für die geplanten Felder wie folgt dar:

- *Index*: Die automatische Durchnummerierung (`AUTO_INCREMENT`) verwendet den Datentyp `INT`, der zur Darstellung ganzer Zahlen dient.
- *Name*: Kurzer Text variabler Länge bis etwa 60 Zeichen – in SQL heißt das `VARCHAR(60)`. Zusätzlich soll ein Index für dieses Feld erstellt werden, um eine eventuelle Suche nach Hotelnamen zu beschleunigen.
- *Stadt*: Bezug auf den numerischen Primärschlüssel der Städte-Tabelle, daher Ganzzahl (`INT`).

- *Preise*: Der Einfachheit halber sind alle Preise in dieser Datenbank ganzzahlig, also INT.
- *Bad*: Eine überschaubare Aufzählung fester Werte, die in SQL ENUM genannt wird.
- *Verpflegung*: Ebenfalls eine festgelegte Aufzählung (ENUM).
- *Postanschrift*: Text variabler Länge bis etwa 100 Zeichen – VARCHAR(100).
- *URL*: Ebenfalls variabler Text – hier genügt erfahrungsgemäß VARCHAR(50).

Die SQL-Abfrage zur Erstellung dieser Tabelle lautet mit den genannten Vorgaben:[1]

```
CREATE TABLE rb_hotels (
  ht_nr INT AUTO_INCREMENT PRIMARY KEY,
  ht_name VARCHAR(60),
  ht_ezpreis INT,
  ht_dzpreis INT,
  ht_bad ENUM('ohne', 'WC', 'Dusche', 'Bad'),
  ht_mahlzeit ENUM('ohne', 'Frühstück', 'HP', 'VP'),
  ht_anschrift VARCHAR(100),
  ht_url VARCHAR(50),
  INDEX (ht_name)
);
```

Wie bereits im Einführungsbeispiel in Kapitel 3 wird auch hier mit Präfixen für die Tabellen- und Feldnamen gearbeitet: Der Name jeder Tabelle in der Datenbank *reisebuero* beginnt mit *rb_*, während die Felder der einzelnen Tabellen jeweils durch ein gemeinsames Kürzel gekennzeichnet werden – hier beispielsweise *ht_* für Hotel. Die Verwendung solcher Präfixe macht immer sofort unmissverständlich klar, zu welcher Datenbank eine Tabelle und zu welcher Tabelle ein Feld gehört. Zudem erspart es Ihnen unter Umständen Schreibarbeit: Im Grunde ist es sinnvoll, Feldern verschiedener Tabellen mit identischer Funktion denselben Namen zu geben. In einer Abfrage müssten diese als *Tabellenname.Feldname* geschrieben werden, da MySQL sonst nicht weiß, welches Feld gemeint ist. Mit Präfixen können Sie dagegen problemlos identische Grundnamen verwenden, weil das Präfix die Felder hinreichend voneinander unterscheidet.

Das Entity-Relationship-Modell

Das *Entity-Relationship-Modell* (kurz *ER-Modell*) ist eine einfache Methode, die in einer Datenbank zu speichernden Gegenstände (*Entities*) und ihre Beziehungen zueinander (*Relationships*) grafisch darzustellen. Die einzelnen Entities werden in einem solchen Modell als Rechtecke dargestellt. In der Regel ergibt sich aus jedem Entity eine Tabelle, in der jedes Element dieses Typs einen Datensatz bildet. Die *Attribute* (Eigenschaften) der Entities, die später die Felder der Tabelle bilden, wer-

1 Sie brauchen dieses Beispiel nicht selbst einzugeben. Weiter unten im Abschnitt »Die Beispieldatenbank« wird die gesamte Datenbank einschließlich dieser Tabelle importiert.

den durch Ovale gekennzeichnet und mit den entsprechenden Entity-Rechtecken verbunden. Eine dritte Art von Information sind die *Rollen*, die als Rauten gezeichnet werden und die Art der Beziehungen zwischen den verschiedenen Entities oder auch Attributen darstellen.

Abbildung 5-1 zeigt ein Entity-Relationship-Modell für das oben erarbeitete Hotel-Beispiel. Informationen über das Entity *Hotel* wie etwa *Preis*, *Badtyp* oder *Verpflegung* sind als Attribute eingezeichnet; in der bereits vorgestellten Tabelle bilden sie die verschiedenen Spalten. Der Primärschlüssel – in diesem Fall die Hotelnummer – ist fett hervorgehoben. Die Attribute des Entity *Stadt* sind in der Abbildung nicht dargestellt, weil sie für den Entwurf der Hotel-Tabelle nicht relevant sind.

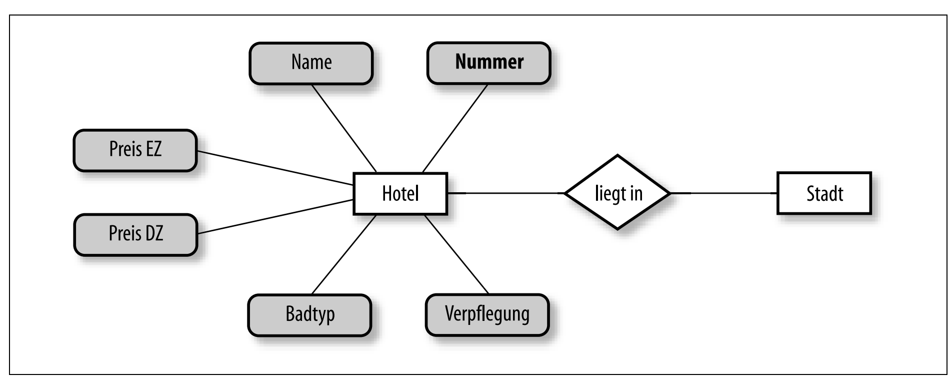


Abbildung 5-1: Einfaches Entity-Relationship-Modell eines Hotel-Entity

Die Rolle, das heißt die Relationsbeschreibung, ist *liegt in*, denn jedes Hotel befindet sich in einer bestimmten Stadt. Da die Verbindungen zwischen Rollen und Entities im ER-Modell keine Richtungsangabe besitzen, ist es empfehlenswert, die Beziehung so darzustellen, dass sie von links nach rechts beziehungsweise von oben nach unten gelesen wird. Falls dies zeichnerisch nicht möglich ist, sollte die Rolle anders formuliert werden – wenn die Stadt sich links vom Hotel befände, hieße eine passende Relation beispielsweise *enthält*. Die Entities können an den Verbindungslinien optional mit *1* beziehungsweise *N* beschriftet werden, um den Relationstyp zu kennzeichnen. Zwischen Städten und Hotels besteht eine *1:n-Relation*: In einer Stadt können beliebig viele Hotels liegen, aber ein einzelnes Hotel kann sich gewiss nicht in mehreren Städten befinden.[2]

Weitere Relationstypen wurden bereits in Kapitel 1 erwähnt: Neben der 1:n-Relation gibt es auch *1:1*- und *m:n-Relationen*. Eine 1:1-Relation besteht, wenn einem Element *genau ein* anderes Element zugeordnet ist. Im Grunde sind Beziehungen zwischen den Attributen eines Entity nichts anderes als 1:1-Relationen. Daher las-

2 Sollte eine Stadtgrenze genau durch ein Hotel verlaufen, wird man sich einigen müssen – allein schon aus steuerlichen Gründen ;-).

sen sich Informationen, die eigentlich dasselbe Entity beschreiben, optional auf mehrere Tabellen verteilen. In der Datenbank *reisebuero* (siehe Abschnitt »Die Beispieldatenbank«) existiert ein solcher Fall: Die Kundendaten wurden auf die beiden Tabellen *rb_kunden* und *rb_kundenkontakte* verteilt, da die Kontaktdetails nicht für jede Operation von Bedeutung sind.

Eine m:n-Relation ordnet beliebig vielen Elementen einer Menge beliebig viele Elemente einer anderen Menge zu. Daraus ergibt sich keine einfache Schlüsselbeziehung, sondern eine neue Tabelle. Abbildung 5-2 zeigt ein vereinfachtes Beispiel: Das Entity *Kunde* kann Reisen buchen, bestehend aus *Flug* und *Hotel*. Jeder einzelne Kunde kann beliebig oft (hintereinander) verreisen; selbstverständlich wird auch jeder Flug und jedes Hotel meist von mehreren Kunden gebucht. Das ergibt m:n-Relationen zwischen *Kunde* und *Flug* beziehungsweise zwischen *Kunde* und *Hotel*. Anders als 1:1- oder 1:n-Relationen sind m:n-Relationen nicht direkt darstellbar. Die Buchungen als solche gehören zu keinem der drei beteiligten Entities, sondern stellen einen separaten Sachverhalt dar. Abbildung 5-3 korrigiert daher das ER-Modell durch die Einführung des neuen Entity *Buchung*. Damit werden die m:n-Relationen in mehrere 1:n-Relationen aufgesplittet. Solche Verfeinerungsprozesse bilden den Hauptnutzen des Entity-Relationship-Modells: An der grafischen Darstellung sind derartige Probleme leichter zu erkennen als in einer Text- oder Tabellenform.

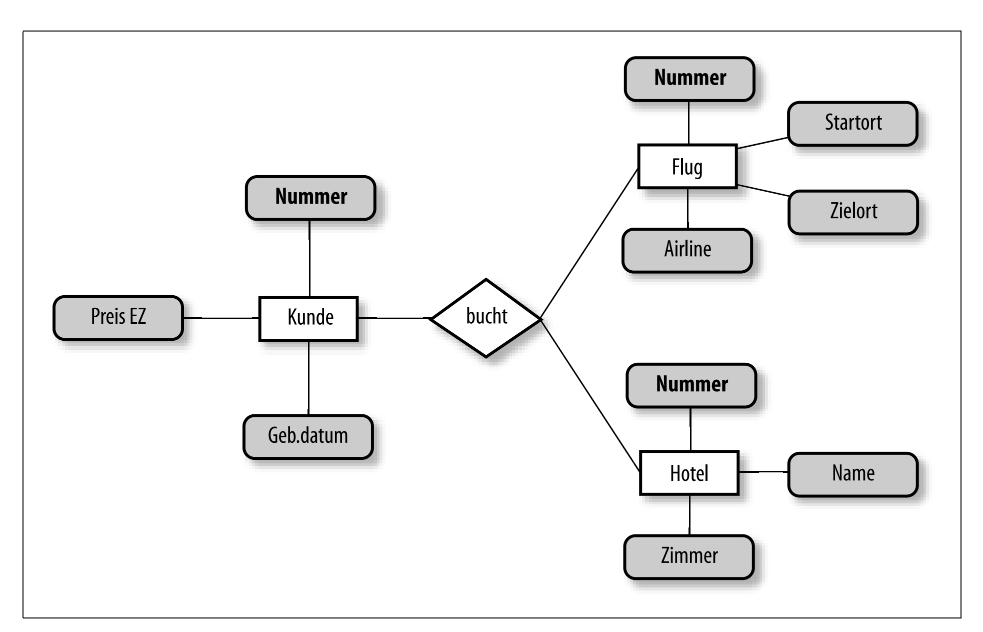


Abbildung 5-2: Entity-Relationship-Modell einer Reisebuchung mit Relationsproblem

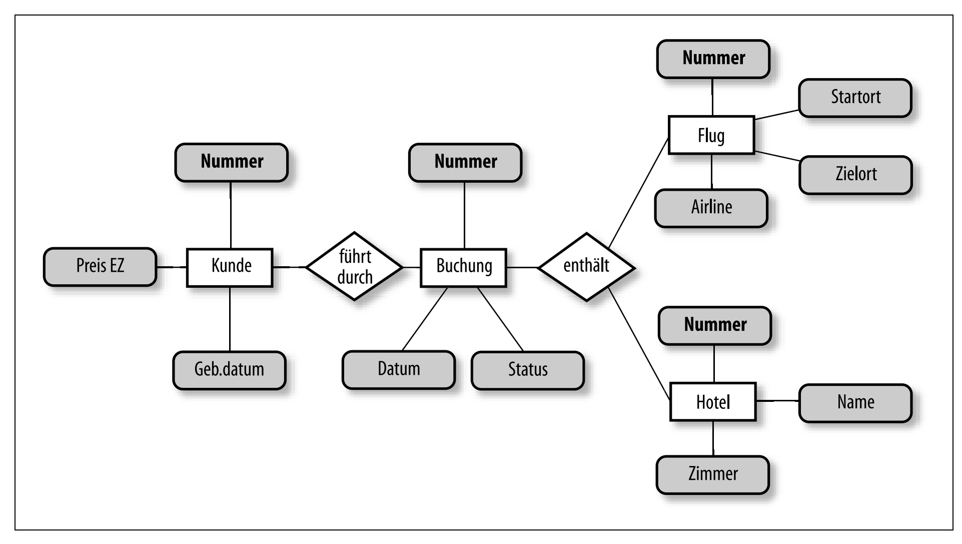


Abbildung 5-3: Entity-Relationship-Modell der Reisebuchung mit aufgelöstem Relationsproblem

Normalisierung

Bereits im ersten Kapitel wurde die Redundanzfreiheit und die daraus resultierende Konsistenz als wichtiges Ziel bei der Modellierung relationaler Datenbanken erwähnt: In einer relationalen Datenbank dürfen Daten nicht mehrfach gespeichert werden, da dies – neben Speichervergeudung – vor allem zu einem Problem führen kann: Verschiedene Exemplare einer gespeicherten Information können durch falsche Eingaben oder unvollständige Aktualisierungen unterschiedliche Werte annehmen; das ist die gefürchtete Inkonsistenz. Deshalb wird in korrekt entworfenen relationalen Datenbanken dafür gesorgt, dass alle Informationen nur je einmal gespeichert und gegebenenfalls durch Schlüssel miteinander verknüpft werden.

Die *Normalisierung* ist eine Art Modellierungsanleitung für relationale Datenbanken, deren strenge Regeln das Auftreten von Inkonsistenzen jeglicher Art verhindern. Es gibt insgesamt sechs sogenannte *Normalformen* (NF), denen Datenbanktabellen entsprechen sollten. Jede dieser Regeln ist strenger als die vorherige; wenn Sie sie der Reihe nach einhalten, erhalten Sie eine korrekt modellierte relationale Datenbank.

Erste Normalform

Die *erste Normalform* (1NF) fordert, dass jedes Feld einer Tabelle *atomar* ist, also eine nicht mehr weiter zerlegbare Einzelinformation enthält. Soll eine Tabelle beispielsweise die Namen von Kunden enthalten, kommt es zu Problemen, wenn Sie Vor- und Nachnamen in einer gemeinsamen Spalte speichern: Wird die natürliche Namensreihenfolge verwendet, also beispielsweise *Peter Schmitz*, kann die Spalte nicht mehr nach dem Nachnamen sortiert werden; die umgekehrte Reihenfolge

Schmitz, Peter verhindert dagegen die normale Verwendung der Namen. Die Lösung liegt natürlich darin, die Namen in zwei verschiedenen Spalten für Vor- und Nachnamen abzulegen.

Beachten Sie, dass Atomarität eine Frage der Perspektive ist – sie hängt davon ab, inwiefern Einzelinformationen in der Praxis benötigt werden. Beispielsweise werden in der Hotel-Tabelle der Reisebüro-Datenbank die Adressen der einzelnen Hotels gespeichert. Sie sollen allerdings nicht dazu verwendet werden, Briefe an diese Hotels zu schreiben, sondern nur als Information auf der Website. Daher genügt es in diesem Fall, die gesamte Anschrift in einer einzelnen Spalte zu speichern. Die Adressen der Kunden sollen dagegen automatisch in Buchungen, Tickets oder Rechnungen eingetragen werden, so dass das Reisebüro hier jeden Bestandteil einzeln braucht, damit die Daten in flexibler Reihenfolge zur Verfügung stehen.

Zweite Normalform

Eine Tabelle ist in der *zweiten Normalform* (2NF), wenn sie in der ersten Normalform ist und wenn zusätzlich jedes Feld vom *gesamten Primärschlüssel* dieser Tabelle abhängt. Datensätze dürfen also nur direkte Informationen über einen einzelnen Sachverhalt (Entity) enthalten, da es andernfalls zu Redundanzen kommen könnte. Betrachten Sie als Beispiel die folgende Tabelle, die Informationen über Städte und in ihnen befindliche Hotels enthält:

```
+----------+------------+------+----------+------------------+
| stadt_id | stadt_name | land | hotel_id | hotel_name       |
+----------+------------+------+----------+------------------+
|        1 | Köln       |    1 |        1 | Bergerhof        |
|        1 | Köln       |    1 |        2 | Hotel Colonia    |
|        2 | Paris      |    2 |        1 | Hotel de la Gare |
|        2 | Paris      |    2 |        2 | Hotel au Jardin  |
+----------+------------+------+----------+------------------+
```

Der Primärschlüssel dieser Tabelle ist kein einzelnes Feld, sondern aus *stadt_id* und *hotel_id* zusammengesetzt. Die Spalte *hotel_name* hängt allerdings nicht vom gesamten Primärschlüssel ab, sondern nur von *hotel_id*. Daher werden Informationen über die Städte unnötigerweise doppelt gespeichert, was zu Inkonsistenzen führen kann. Die Lösung besteht in der Aufteilung der Informationen auf zwei Tabellen, wobei der Bezug auf die Stadt als Fremdschlüssel in der Hotel-Tabelle gespeichert wird:

```
+----------+------------+------+
| stadt_id | stadt_name | land |
+----------+------------+------+
|        1 | Köln       |    1 |
|        2 | Paris      |    2 |
+----------+------------+------+
```

```
+----------+-----------------+-------+
| hotel_id | hotel_name      | stadt |
+----------+-----------------+-------+
|        1 | Bergerhof       |     1 |
|        2 | Hotel Colonia   |     1 |
|        3 | Hotel de la Gare |    2 |
|        4 | Hotel au Jardin |     2 |
+----------+-----------------+-------+
```

Dritte Normalform

Die *dritte Normalform* (3NF) gilt, wenn die zweite Normalform erfüllt ist und alle Felder voneinander *funktional unabhängig* sind. Eine Tabelle verletzt die dritte Normalform, wenn sie eine eindeutig zu einem einzelnen Feld gehörige Zusatzinformation enthält, obwohl dieses Feld nicht den Schlüssel bildet. Das folgende Beispiel zeigt eine Tabelle von Fluggesellschaften, in der gleichzeitig die jeweilige Hauptstadt des Herkunftslands angegeben wird:

```
+----+--------------------+----------------+------------+
| id | name               | land           | hauptstadt |
+----+--------------------+----------------+------------+
|  1 | Air France         | Frankreich     | Paris      |
|  2 | Austrian Airlines  | Österreich     | Wien       |
|  3 | British Airways    | Großbritannien | London     |
|  4 | Germanwings        | Deutschland    | Berlin     |
+----+--------------------+----------------+------------+
```

Die Hauptstadt kennzeichnet jeweils das Land, aber nicht die Fluggesellschaft, um die es in dieser Tabelle eigentlich geht. Die Lösung liegt wiederum in einer Aufteilung der Informationen auf zwei Tabellen:

```
+----+----------------+------------+
| id | land           | hauptstadt |
+----+----------------+------------+
|  1 | Deutschland    | Berlin     |
|  2 | Frankreich     | Paris      |
|  3 | Großbritannien | London     |
|  4 | Österreich     | Wien       |
+----+----------------+------------+

+----+--------------------+------+
| id | name               | land |
+----+--------------------+------+
|  1 | Air France         |    2 |
|  2 | Austrian Airlines  |    4 |
|  3 | British Airways    |    3 |
|  4 | Germanwings        |    1 |
+----+--------------------+------+
```

Boyce-Codd-Normalform

Es existiert noch eine strengere Form der dritten Normalform: die *Boyce-Codd-Normalform* (BCNF). Eine Tabelle mit einem Primärschlüssel, der aus mehreren Feldern zusammengesetzt ist, kann die BCNF verletzen, wenn der Wert eines Felds nicht vom gesamten Primärschlüssel, sondern nur von einem seiner Felder abhängt. Die Boyce-Codd-Normalform lässt sich herstellen, indem diese Tabelle in zwei einzelne Tabellen aufgeteilt wird, in denen jeweils eines der Felder den Primärschlüssel bildet.

Vierte Normalform

Die *vierte Normalform* (4NF) kümmert sich um *mehrwertige Abhängigkeiten* (multivalued dependencies), bei denen eine Beziehung zwischen verschiedenen Informationen nicht so in zwei Tabellen unterteilt werden kann, dass eine 1:n- oder die umgekehrte n:1-Relation entsteht.

Ein Beispiel: In einer zweispaltigen Tabelle werden durch einen Fremdschlüssel Personen referenziert und in der zweiten Spalte durch einen weiteren Fremdschlüssel die von diesen Personen gebuchten Flüge. Da eine Person mehrere Flüge buchen kann, kann sowohl jede Person als auch jeder Flug mehrmals vorkommen. Die folgende Tabelle konkretisiert den Sachverhalt, indem sie die Schlüsselnummern durch Textwerte darstellt:

```
+---------+---------+--------------------+
| buch_nr | person  | flug               |
+---------+---------+--------------------+
|       1 | Schmitz | Köln-Paris         |
|       2 | Müller  | Köln-Paris         |
|       3 | Schmitz | Köln-London        |
|       4 | Huber   | Frankfurt-Madrid   |
+---------+---------+--------------------+
```

Angenommen, es kommt eine weitere Spalte mit Extras einzelner Reisebuchungen hinzu, etwa ein Spezialprogramm für Kinder oder ein Sonderrabatt. Auch diese beziehen sich auf eine separate Tabelle, und jede Buchung kann mehrere Extras enthalten. Das kann zu einer doppelten Nennung einzelner Flugbuchungen führen:

```
+---------+---------+--------------------+-----------+
| buch_nr | person  | flug               | extra     |
+---------+---------+--------------------+-----------+
|       1 | Schmitz | Köln-Paris         | Kinderpr. |
|       1 | Schmitz | Köln-Paris         | Rabatt    |
|       2 | Müller  | Köln-Paris         | Mietwagen |
|       2 | Müller  | Köln-Paris         | Rabatt    |
|       3 | Schmitz | Köln-London        | Kinderpr. |
|       4 | Huber   | Frankfurt-Madrid   | Rabatt    |
|       4 | Huber   | Frankfurt-Madrid   | Kulturpr. |
+---------+---------+--------------------+-----------+
```

Auch in diesem Fall schafft eine Aufteilung auf mehrere Tabellen Abhilfe (hier wieder mit der expliziten Nennung der Informationen statt der Verwendung numerischer Schlüssel – eigentlich beziehen sich die Felder *flug* beziehungsweise *extra* auf separate Tabellen):

```
+---------+---------+--------------------+
| buch_nr | person  | flug               |
+---------+---------+--------------------+
|       1 | Schmitz | Köln-Paris         |
|       2 | Müller  | Köln-Paris         |
|       3 | Schmitz | Köln-London        |
|       4 | Huber   | Frankfurt-Madrid   |
+---------+---------+--------------------+

+---------+----------------+
| buch-nr | extra          |
+---------+----------------+
|       1 | Kinderprogramm |
|       1 | Rabatt         |
|       2 | Mietwagen      |
|       2 | Rabatt         |
|       3 | Kinderprogramm |
|       4 | Rabatt         |
|       4 | Kulturprogramm |
+---------+----------------+
```

Fünfte Normalform

Die *fünfte Normalform* (5NF) schließlich ist erfüllt, wenn innerhalb einer Tabelle nur *triviale Join-Abhängigkeiten* existieren. Join-Abhängigkeiten bestehen in jeder Tabelle, solange sie sich in mehrere einzelne Tabellen mit demselben Schlüssel unterteilen lässt. Trivial sind diese Abhängigkeiten, wenn eine Verknüpfung zweier solcher Einzeltabellen keine Redundanz durch einen verdoppelten Datensatz ergäbe. In der Praxis ist diese Normalform also nur von Bedeutung, wenn zwei bisher getrennte Tabellen zusammengefügt werden sollen.

Im Reisebüro-Beispiel könnte die Tabelle mit den Basisdaten der Kunden problemlos mit den Kundenkontaktdaten verknüpft werden, da sich beide auf das Entity Kunde beziehen und pro Kunde je genau einen Datensatz enthalten.



In der Praxis kommt es mitunter vor, dass bewusst gegen einige der Normalisierungsregeln verstoßen wird. Solange Sie genau wissen, was Sie tun, kann eine solche »Denormalisierung« manchmal die Verarbeitungsgeschwindigkeit Ihrer Datenbanken verbessern oder Ihnen die Arbeit erleichtern. In diesem Buch werden Sie dergleichen allerdings nicht antreffen.

Wenn relationale Datenbanken überfordert sind



Manche komplexen Datenstrukturen lassen sich mithilfe des relationalen Modells nur schlecht darstellen. Das neuere *objektorientierte* Datenbankmodell ermöglicht dagegen beliebige, frei verknüpfte Datenstrukturen: Objektorientierte Datenbanken enthalten Klassen und Objekte, die genauso aufgebaut sind wie in objektorientierten Programmiersprachen.

Ein Beispiel für Daten, für die das relationale Datenbankmodell wenig geeignet ist, sind Entfernungsangaben zwischen Orten. Tabelle 5-1 versucht, diese Informationen relational zu speichern.

Tabelle 5-1: Eine für das relationale Modell wenig geeignete Datenbanktabelle

Startort	Zielort	Entfernung
Köln	Paris	490
Köln	Rom	1400
Köln	Istanbul	2400
Paris	Rom	1410
Paris	Istanbul	2740
Rom	Istanbul	1450

Zwei Spalten in der Tabelle enthalten Ortsangaben, also gleichartige Daten. Kein logisches Kriterium kann bestimmen, welche Städte als *Startort* und welche als *Zielort* eingetragen werden. Auch die Darstellung der Entfernungen in beide Richtungen wäre keine Lösung, da sie Redundanzen ergäbe, die zu Inkonsistenzen führen können.[a]

Für die objektorientierte Modellierung sind diese Entfernungen dagegen ohne Schwierigkeiten darstellbar: Eine Klasse namens *Ort* enthält eine Liste von Zielen, zu denen direkte Verbindungen bestehen. Die Ziele sind nichts weiter als Verweise auf andere Elemente vom Typ *Ort*.

In der *Object Definition Language* (ODL), einer Art standardisierter objektorientierter Modellierungssprache, bilden die durch das Schlüsselwort `class` gekennzeichneten Klassen die übergeordnete Datenstruktur. *Attribute* (vergleichbar mit den Spalten einer relationalen Datenbanktabelle) werden durch das Schlüsselwort `attribute`, einen Datentyp und eine Bezeichnung definiert. Eine ODL-Definition der Entfernungstabelle könnte beispielsweise folgendermaßen aussehen:

```
class Ort {
   attribute string Name;
   struct Entfernung {
      relationship Ort Zielort;
      attribute short Kilometer;
   };
   array (struct Entfernung) Entfernungen;
}
```

→

Das Schlüsselwort `array` bezeichnet eine Liste mehrerer Elemente desselben Datentyps, genau wie in vielen Programmiersprachen. Hier wird eine Entfernung als Struktur aus einer Beziehung (`relationship`) zu einem anderen Ort und der zugehörigen Kilometeranzahl gebildet. Die Entfernungen selbst werden in einem Array aus Elementen dieser Datenstruktur dargestellt.

Neben den rein objektorientierten Datenbanken existieren auch Mischtypen zwischen relationalen und objektorientierten Techniken. Eines der am weitesten verbreiteten Datenbanksysteme mit objektrelationalen Funktionen ist die Open Source-Datenbank *PostgreSQL*.

[a] Sehr elegant lässt sich dieses Problem übrigens lösen, indem man die geografischen Koordinaten der Orte speichert und die Entfernungen einfach berechnet. In MySQL sind diese sogenannten GIS-Features (Geospacial Information System) recht neu; in Anhang D finden Sie eine Web-URL zu diesem Thema.

Die Beispieldatenbank

Die Datenbank *reisebuero* besteht aus elf Tabellen. Der Umfang dieser Datenbank ist angesichts der Leistungsfähigkeit neuerer MySQL-Versionen nicht der Rede wert, aber doch zu groß für einen vollständigen Abdruck mitsamt Inhalten. Deshalb wird im Folgenden zwar die Struktur, aber nicht der komplette Inhalt der Tabellen erläutert.

Um mit diesem Kapitel und dem Rest dieses Buchs arbeiten zu können, muss diese Datenbank zunächst auf Ihrem System eingerichtet werden. Dazu dient die Datei *reisebuero.sql* im Verzeichnis *database* auf der beiliegenden CD-ROM. Öffnen Sie zum Import ein Terminal (Linux) beziehungsweise die Eingabeaufforderung (Windows). Wechseln Sie per `cd`-Kommando in das Verzeichnis *database* der CD-ROM. Starten Sie anschließend den Kommandozeilenclient mysql als *root* und geben Sie folgendes Kommando ein:

```
mysql> \. reisebuero.sql
```

Nun laufen eine Weile lang Meldungen an Ihnen vorbei, anschließend steht die Datenbank auf Ihrem System zur Verfügung.



Sollte Ihr MySQL-Server bereits eine Datenbank mit dem Namen *reisebuero* enthalten, wird diese automatisch überschrieben, damit die anfängliche `CREATE DATABASE`-Abfrage nicht fehlschlägt. Eine vorhandene Datenbank dieses Namens müssten Sie also zunächst umbenennen.

Struktur der Datenbank

In der Datenbank des Reisebüros müssen zahlreiche verschiedene Daten gespeichert werden. Insgesamt geht es um die angebotenen Reisen, bestehend aus Flug- und Hotelinformationen, um die bereisten Städte und Länder sowie – natürlich –

um die Kunden, die sich für eine Reise interessieren oder gar eine buchen möchten. Diese oft aus mehreren Tabellen bestehenden Datensammlungen stehen zunächst relativ beziehungslos nebeneinander. Erst die Tabelle *rb_buchungen* nimmt auf viele von ihnen Bezug. Nicht unmittelbar zum Kern der Buchungsangelegenheiten gehört zuletzt eine Tabelle über Sehenswürdigkeiten, die den potenziellen Kunden den Besuch der einzelnen Städte schmackhaft machen sollen.

Tabelle 5-2 zeigt zunächst eine Übersicht über sämtliche Tabellen der Datenbank und ihre Aufgaben.

Tabelle 5-2: Die Tabellen der Datenbank »reisebuero«

Tabelle	Funktion
rb_laender	Länder, in die Reisen angeboten werden
rb_staedte	Städte, in die Reisen angeboten werden
rb_airlines	Fluggesellschaften
rb_airports	Flughäfen
rb_flugstrecken	Flugangebote der Fluggesellschaften
rb_fluege	konkrete Flüge mit Datum
rb_hotels	die verfügbaren Hotels
rb_kunden	Basisdaten der Kunden
rb_kundenkontakte	Kontaktdaten der Kunden
rb_buchungen	konkrete Reisebuchungen
rb_sehensw	Hauptsehenswürdigkeiten der einzelnen Städte

In der recht umfangreichen Tabelle 5-3 werden die verschiedenen Spalten der einzelnen Datenbanktabellen vorgestellt.

Tabelle 5-3: Bedeutung der Spalten aller Tabellen der Datenbank »reisebuero«

Spalte	Datentyp	Optionen	Bedeutung
Tabelle rb_laender			
la_nr	`INT`	Primärschlüssel, `AUTO_INCREMENT`	
la_name	`VARCHAR(40)`	Index	deutscher Name des Landes
la_eigen	`VARCHAR(40)`	Index	Eigenname des Landes
la_zkuerzel	`CHAR(3)`		Kfz-Kennzeichen des Landes (z.B. *D*)
la_dkuerzel	`CHAR(2)`		Top-Level-Domain (z.B. *de*)
la_bemerk	`VARCHAR(100)`		Bemerkungen (z.B. Einreisebestimmungen)
Tabelle rb_staedte			
st_nr	`INT`	Primärschlüssel, `AUTO_INCREMENT`	

Tabelle 5-3: Bedeutung der Spalten aller Tabellen der Datenbank »reisebuero« (Fortsetzung)

Spalte	Datentyp	Optionen	Bedeutung
st_name	VARCHAR(40)	Index	deutscher Name der Stadt
st_eigen	VARCHAR(40)	Index	Eigenname der Stadt
st_land	INT		Land (Bezug auf *rb_laender.la_nr*)
st_bild_url	VARCHAR(40)		URL/Dateiname einer Abbildung
st_text_url	VARCHAR(50)		URL/Dateiname einer Beschreibung
Tabelle rb_airlines			
ai_nr	INT	Primärschlüssel, AUTO_INCREMENT	
ai_name	VARCHAR(40)	Index	Name der Fluggesellschaft
ai_kuerzel	CHAR(6)		Kürzel (z.B. *LH* für Lufthansa)
ai_land	INT		Herkunftsland (Bezug auf *rb_laender.la_nr*)
ai_url	VARCHAR(50)		Website der Fluggesellschaft
Tabelle rb_airports			
ap_nr	INT	Primärschlüssel, AUTO_INCREMENT	
ap_name	VARCHAR(40)	Index	Name des Flughafens
ap_zusatz	VARCHAR(40)		Namenszusatz (z.B. Istanbul -> Atatürk Intl. Airport)
ap_kuerzel	CHAR(6)		internationales Kürzel (z.B. *CGN* für Köln/Bonn)
ap_stadt	INT		Stadt (Bezug auf *rb_staedte.st_nr*)
ap_url	VARCHAR(50)		Website des Flughafens
Tabelle rb_flugstrecken			
fs_nr	INT	Primärschlüssel, AUTO_INCREMENT	
fs_airline	INT		Bezug auf *rb_airlines.ai_nr*
fs_onr	CHAR(10)		offizielle Flugnummer
fs_start	INT		Bezug auf *rb_airports.ap_nr*
fs_ziel	INT		Bezug auf *rb_airports.ap_nr*
fs_dauer	INT		Flugdauer in Minuten
Tabelle rb_fluege			
fl_nr	INT	Primärschlüssel, AUTO_INCREMENT	
fl_strecke	INT		Bezug auf *rb_flugstrecken.fs_nr*
fl_preis	INT		Flugpreis

Tabelle 5-3: Bedeutung der Spalten aller Tabellen der Datenbank »reisebuero« (Fortsetzung)

Spalte	Datentyp	Optionen	Bedeutung
fl_datum	DATE		Abflugdatum
fl_zeit	DATE		Abfluguhrzeit
Tabelle rb_hotels			
ht_nr	INT	Primärschlüssel, AUTO_INCREMENT	
ht_name	VARCHAR(60)	Index	Name des Hotels
ht_stadt	INT		Bezug auf *rb_staedte.st_nr*
ht_ezpreis	INT		Preis/Einzelzimmer
ht_dzpreis	INT		Preis/Doppelzimmer
ht_bad	ENUM	('ohne', 'WC', 'Dusche', 'Bad')	WC/Bad-Varianten
ht_mahlzeit	ENUM	('ohne', 'Frühstück', 'HP', 'VP')	Inklusiv-Mahlzeiten
ht_anschrift	VARCHAR(100)		Adresse des Hotels
ht_url	VARCHAR(50)		Website des Hotels
Tabelle rb_kunden			
kd_nr	INT	Primärschlüssel, AUTO_INCREMENT	
kd_mail	VARCHAR(50)		E-Mail-Adresse
kd_name	VARCHAR(40)	Index	Nachname
kd_vorname	VARCHAR(40)		Vorname
kd_gdatum	DATE		Geburtsdatum
kd_bemerk	VARCHAR(100)		Anmerkungen zum Kunden
Tabelle rb_kundenkontakte			
kk_nr	INT		Bezug auf *rb_kunden.kd_nr*
kk_strasse	VARCHAR(50)		Straße
kk_hausnr	VARCHAR(10)		Hausnummer
kk_plz	VARCHAR(10)		Postleitzahl
kk_ort	VARCHAR(50)	Index	Wohnort
kk_land	INT		Bezug auf *rb_laender.la_nr*
kk_tel	VARCHAR(30)		Telefonnummer
Tabelle rb_buchungen			
bu_nr	INT	Primärschlüssel, AUTO_INCREMENT	
bu_datum	DATE		Buchungsdatum
bu_kunde	INT		Bezug auf *rb_kunden.kd_nr*
bu_perszahl	INT		Anzahl Personen

Tabelle 5-3: Bedeutung der Spalten aller Tabellen der Datenbank »reisebuero« (Fortsetzung)

Spalte	Datentyp	Optionen	Bedeutung
bu_startdat	DATE		Hinreisedatum
bu_enddat	DATE		Rückreisedatum
bu_hinflug	INT		Bezug auf *rb_fluege.fl_nr*
bu_rueckflug	INT		Bezug auf *rb_fluege.fl_nr*
bu_hotel	INT		Bezug auf *rb_hotels.ht_nr*
bu_ezanz	INT		Anzahl Einzelzimmer
bu_dzanz	INT		Anzahl Doppelzimmer
bu_status	ENUM	('in_arbeit', 'aktiv', 'storniert')	Buchungsstatus
Tabelle rb_sehensw			
sw_nr	INT	Primärschlüssel, AUTO_INCREMENT	
sw_stadt	INT		Bezug auf *rb_staedte.st_nr*
sw_name	VARCHAR(50)	Index	deutscher Name der Sehenswürdigkeit
sw_eigen	VARCHAR(50)	Index	Eigenname der Sehenswürdigkeit
sw_bild_url	VARCHAR(40)		URL/Dateiname einer Abbildung
sw_anschrift	VARCHAR(100)		Adresse der Sehenswürdigkeit
sw_beschr	TEXT		Kurzbeschreibung der Sehenswürdigkeit

Datenbanken und Tabellen erstellen

Bereits in Kapitel 3 wurde kurz besprochen, welche SQL-Abfragen beim Erzeugen neuer Datenbanken und Tabellen zum Einsatz kommen. In diesem Abschnitt werden alle wichtigen Optionen für diese Abfragen vorgestellt.

Erstellungsabfragen

Zur Erzeugung von Datenbanken und Tabellen werden spezielle SQL-Abfragen verwendet, die mit `CREATE DATABASE` beziehungsweise `CREATE TABLE` beginnen. Da Sie die Datenbank *reisebuero* bereits in Ihren Datenbankserver importiert haben sollten, brauchen Sie die Beispiele in diesem Abschnitt nicht einzutippen. Wenn Sie es zu Übungszwecken dennoch tun möchten, sollten Sie zunächst eine zusätzliche Datenbank dafür erstellen.

`CREATE DATABASE` erstellt eine neue Datenbank und besitzt nicht viele Optionen. In aller Regel wird lediglich der Name der Datenbank angegeben, zum Beispiel:

```
CREATE DATABASE reisebuero;
```

Optional können Sie mithilfe der Option [DEFAULT] CHARACTER SET oder kurz CHARSET einen Zeichensatz und mittels COLLATE eine Sortierreihenfolge angeben. Für Deutsch bieten sich der Zeichensatz *latin1* (ohnehin Standard) und die Kollation *latin1_german1_ci* (Sortierung in Wörterbuchreihenfolge) an. Die ganze Abfrage sieht in diesem Fall folgendermaßen aus:

```
CREATE DATABASE reisebuero CHARACTER SET latin1
COLLATE latin1_german1_ci;
```

Zeichensatz und Kollation gelten zunächst einmal automatisch für jede Tabelle und jedes (Text-)Datenfeld in der Datenbank, es sei denn, Sie stellen auf einer dieser untergeordneten Ebenen ausdrücklich einen anderen Wert ein.

Eine letzte zusätzliche Möglichkeit für die Tabellenerstellung bietet die Klausel IF NOT EXISTS: Die Datenbank wird nur dann angelegt, falls noch keine gleichnamige Datenbank existiert. Die folgende Abfrage können Sie also in jedem Fall fehler- und gefahrenfrei ausführen:

```
CREATE DATABASE IF NOT EXISTS reisebuero;
```

CREATE TABLE-Abfragen zur Erzeugung von Tabellen sind dagegen komplexer. Die allgemeine Grundform dieser SQL-Anweisung sieht folgendermaßen aus:

```
CREATE TABLE Tabellenname (
   Feldname1 Datentyp [Optionen]
[, Feldname2 Datentyp [Optionen], ...]
);
```

Hier sehen Sie als kurzes Beispiel die Erstellungsabfrage für die Tabelle *rb_kunden*:

```
CREATE TABLE rb_kunden (
   kd_nr INT AUTO_INCREMENT PRIMARY KEY,
   kd_name VARCHAR(40),
   kd_vorname VARCHAR(40),
   kd_gdatum DATE,
   kd_bemerk VARCHAR(100),
);
```

Der für den Nachnamen geplante Index wurde hier noch nicht hinzugefügt. Schlüssel und Indizes werden weiter unten in einem eigenen Abschnitt behandelt.

Auch die Erzeugung einer Tabelle können Sie mittels IF NOT EXISTS davon abhängig machen, ob sie bereits existiert oder nicht. Ein verkürztes Beispiel:

```
CREATE TABLE IF NOT EXISTS rb_kunden (...);
```

Eine wichtige Option ist der Tabellentyp (*Storage Engine*). Seit Version 3.23 können MySQL-Datenbanken Tabellen enthalten, die auf unterschiedliche Art und Weise gespeichert werden und verschiedene Features unterstützen. Der Typ wird durch die Option ENGINE=*Typname* oder TYPE=*Typname* angegeben. Die wichtigsten Typen sind folgende:

- *MyISAM*: Eine Weiterentwicklung des klassischen MySQL-Tabellentyps *ISAM* (Indexed Sequential Access Method). Die Verarbeitung von MyISAM-Tabellen erfolgt schneller als die anderer permanent gespeicherter Tabellentypen, außerdem bieten sie die meisten Indexoptionen (siehe unten) sowie Unterstützung für Geodaten. Der Nachteil: MyISAM ist nicht transaktionsfähig. Wenn Sie keine Engine angeben, ist MyISAM der Standardtyp (außer unter Windows).
- *InnoDB*: Die Storage-Engine InnoDB wird von der Firma Innobase Oy in Finnland (*http://www.innobase.com*) entwickelt. Diese Firma wurde pikanterweise von MySQLs kommerziellem Konkurrenten Oracle aufgekauft, aber bisher bleibt die Engine in MySQL integriert.[3] Genau wie MySQL selbst ist sie unter zwei verschiedenen Lizenzen verfügbar: unter der GNU GPL für die einfache Datenbankanbindung und die Einbettung in Open Source-Projekte sowie unter einer kommerziellen Lizenz für die Einbindung in kommerzielle Produkte. InnoDB ist für Transaktionen optimiert, unterstützt aber nicht alle Indexarten – insbesondere keine Volltextindizes. InnoDB ist seit MySQL 4.1.5 der Standardtyp auf Windows-Systemen.
- *BDB* oder *Berkeley DB*: Berkeley DB ist seit Jahren auch unabhängig von MySQL ein beliebtes Open Source-Datenformat für datenbankähnliche Strukturen. Die Storage-Engine wird von Sleepycat Software (*http://www.sleepycat.com*) gepflegt. MySQL verwendet eine spezielle, angepasste Version; die gängigen Berkeley DB-Pakete in Linux-Distributionen oder anderen Unix-Systemen arbeiten nicht mit MySQL zusammen. Genau wie InnoDB unterstützt auch BDB Transaktionen. Beachten Sie, dass Berkeley DB nur in manchen Linux- und Unix-Versionen verfügbar ist, nicht aber unter Windows und Mac OS X.
- *MEMORY* oder *HEAP*: Die Tabelle wird im Arbeitsspeicher angelegt, geht also beim Beenden des MySQL-Servers verloren. Dieser Typ bietet die beste Performance, ist aber naturgemäß nur für temporäre Arbeitstabellen geeignet. Indizes werden unterstützt, aber keine Transaktionen.
- *CSV*: Die Daten werden in einer Textdatei als *Comma-Separated Values* gespeichert, also etwa in folgendem Format:

  ```
  "1", "Schmitz", "Klaus", "1972-09-17", ""
  "2", "Becker", "Hans", "1956-10-14", ""
  ```

  Der Vorteil dieses Tabellentyps besteht darin, dass er sich leicht in Tabellenkalkulationen oder selbst geschriebene Programme importieren lässt. In der Regel überwiegen allerdings die Nachteile: In CSV-Tabellen können weder Indizes noch Transaktionen eingesetzt werden, und besonders schnell ist diese Storage-Engine auch nicht.

3 Die MySQL-Entwickler arbeiten zur zeit an einer eigenen transaktionsfähigen Engine namens Falcon, die in der kommenden MySQL-Version 6.0 enthalten sein soll.

- *ARCHIVE*: Diese Engine speichert die Tabellendaten komprimiert. Dies vermindert den Speicherbedarf, verschlechtert aber dafür die Performance ein wenig, weil die Daten bei jedem Zugriff entpackt werden müssen. Die wichtigste Einschränkung besteht darin, dass Sie nur Datentypen mit festgelegter Länge einsetzen dürfen – beispielsweise nur CHAR, aber keinen VARCHAR.
- *FEDERATED*: Der Tabellentyp FEDERATED speichert Tabellendaten nicht im lokalen MySQL-Server, sondern dient dem Zugriff auf Tabellen mit identischem Format auf einem anderen Server. Auf diese Weise können Sie ohne Programmierschnittstelle auf entfernte Datenbanken zugreifen. Angenommen, auf Server 1 mit der IP-Adresse 192.168.0.2 befindet sich die Tabellendefinition von *rb_sehensw* in der Datenbank *reisebuero*:

  ```
  CREATE TABLE rb_sehensw (
    sw_nr INT AUTO_INCREMENT PRIMARY KEY,
    sw_stadt INT DEFAULT '0',
    sw_name VARCHAR(50),
    sw_eigen VARCHAR(50),
    sw_bild_url VARCHAR(40),
    sw_anschrift VARCHAR(100),
    sw_beschr TEXT,
  ) ENGINE=MyISAM DEFAULT CHARSET=latin1 COLLATE=latin1_german1_ci;
  ```

  Wenn Server 2 diese Tabelle per Federated-Verbindung nutzen soll, müssen Sie dort folgende Tabellendefinition erzeugen:

  ```
  CREATE TABLE rb_sehensw (
    sw_nr INT AUTO_INCREMENT PRIMARY KEY,
    sw_stadt INT DEFAULT '0',
    sw_name VARCHAR(50),
    sw_eigen VARCHAR(50),
    sw_bild_url VARCHAR(40),
    sw_anschrift VARCHAR(100),
    sw_beschr TEXT,
  ) ENGINE=FEDERATED
    CONNECTION='mysql://root@192.168.0.2/reisebuero/rb_sehensw';
  ```

  Das Schlüsselwort CONNECTION gibt es erst seit MySQL 5.0.13; in älteren Versionen müssen Sie die Daten in den COMMENT schreiben. Vor dem Hostnamen oder der IP-Adresse des entfernten Datenbankhosts muss ein Username eingetragen werden, der auf die betreffende Tabelle zugreifen darf. Wenn der angegebene User nicht bereits angemeldet ist, müssen Sie auch sein Passwort angeben, indem Sie `User:Passwort@Datenbankhost` (zum Beispiel `root:geheim@192.168.0.2`) schreiben.

  Verwechseln Sie dieses Konzept nicht mit der in Kapitel 9 vorgestellten Replikation, die tatsächlich lokale Livekopien entfernter Datenbanken erzeugt.
- *BLACKHOLE*: Diese spezielle Engine verwirft sämtliche Daten, die in ihren Tabellen gespeichert werden.

Das folgende (gekürzte) Beispiel erstellt die Tabelle *rb_kunden* als InnoDB-Tabelle:

```
CREATE TABLE rb_kunden (...) ENGINE=InnoDB;
```

Auch für einzelne Tabellen können Sie Zeichensätze und Sortierreihenfolgen angeben. Das geschieht wiederum mithilfe der Optionen `[DEFAULT] CHARACTER SET` beziehungsweise `COLLATE`. Das nachfolgende Beispiel stellt den Standardzeichensatz *latin1* und die Sortierreihenfolge *Deutsch (Telefonbuch)* ein:

```
CREATE TABLE rb_kunden (...)
DEFAULT CHARACTER SET latin1
COLLATE latin1_german2_ci;
```

Sogar für einzelne Spalten ist die Angabe alternativer Zeichensätze beziehungsweise Kollationen möglich. Dies ist beispielsweise nützlich, wenn Sie eine wörterbuchartige Tabelle anlegen, in der Sprachen mit verschiedenen Schriften vorkommen. Das folgende Beispiel wäre die passende Struktur für eine (in diesem Buch nicht weiter ausgeführte) Zusatztabelle in der Datenbank *reisebuero*, die wichtige Begriffe und Phrasen für das Überleben von Reisenden in den Sprachen der beliebtesten Reiseländer enthält:

```
CREATE TABLE rb_sprachfuehrer (
  sp_nr INT AUTO_INCREMENT PRIMARY KEY,
  sp_deutsch VARCHAR(40) COLLATE latin1_german1_ci,
  sp_englisch VARCHAR(40) COLLATE latin1_general_ci,
  sp_italienisch VARCHAR(40) COLLATE latin1_general_ci,
  sp_spanisch VARCHAR(40) COLLATE latin1_spanish_ci,
  sp_tuerkisch VARCHAR(40) CHARSET latin5 COLLATE latin5_turkish_ci,
  sp_griechisch VARCHAR(40) CHARSET greek COLLATE greek_general_ci
) CHARSET latin1;
```

Eine spezielle Form der Tabellenerstellungsabfrage bietet die `LIKE`-Klausel: Sie kopiert die Struktur einer vorhandenen Tabelle. Die allgemeine Syntax sieht folgendermaßen aus:

```
CREATE TABLE Tabellenname LIKE [Datenbankname.]Tabellenname;
```

Hier ein Beispiel, das die Struktur von *rb_hotels* übernimmt:

```
CREATE TABLE rb_pensionen LIKE rb_hotels;
```

Wenn Sie anschließend auch die Daten aus der ursprünglichen Tabelle übernehmen möchten, funktioniert dies so:

```
INSERT INTO rb_pensionen SELECT * FROM rb_hotels;
```

Noch einen Schritt weiter geht eine Kombination aus `CREATE TABLE` und `SELECT`: Diese SQL-Anweisung erstellt eine neue Tabelle mit der Struktur der Auswahlabfrage und kopiert sämtliche gelesenen Datensätze hinein. Das folgende Beispiel erzeugt eine vollständige Kopie von *rb_kunden*:

```
CREATE TABLE rb_kunden_kopie SELECT * FROM rb_kunden;
```



Bei einer auf diese Weise erzeugten Kopie werden AUTO_INCREMENT-Einstellungen und manche Indizes nicht übernommen. Zudem werden manche Datentypen falsch kopiert. Überprüfen Sie das Ergebnis auf jeden Fall ganz genau.

Sie können auf diese Weise nicht nur bestehende Tabellen kopieren, sondern auch beliebige Abfrageergebnisse als neue Tabellen anlegen. Näheres über die vielfältigen Möglichkeiten von SELECT-Abfragen erfahren Sie im nächsten Kapitel.

Zur Tabellenerstellung gehören auch die MySQL-Datentypen sowie Schlüssel und Indizes; sie werden in den nächsten beiden Abschnitten behandelt. Daneben gibt es noch weitere Tabellenoptionen, die aber nicht ganz so wichtig sind. Die vollständige Liste finden Sie im Abschnitt 13.1.5 der offiziellen MySQL-Dokumentation (Verzeichnis *docs* auf der beiliegenden CD).

MySQL-Datentypen

Die zulässigen Datentypen für Felder in MySQL-Tabellen entsprechen bis auf wenige Ausnahmen dem SQL-Standard. Es gibt folgende Gruppen von Datentypen:

- ganzzahlige Typen
- Fließkommatypen
- Datum und Uhrzeit
- Zeichenketten
- Text- und Binärdatenblöcke
- Aufzählungstypen
- geografische/geometrische Typen (GIS) – dieses Thema geht über den Umfang des vorliegenden Buchs hinaus; in Anhang D finden Sie die URL einer Website zu diesem Thema

Ganzzahlige Typen

Der Standardtyp für ganzzahlige Werte ist INT. Es handelt sich um einen 32-Bit-Integer, der Werte zwischen –2.147.483.648 und +2.147.483.647 annehmen kann. Mit der speziellen Option UNSIGNED (vorzeichenlos) sind Werte von 0 bis 4.294.967.295 möglich. Neben INT definiert MySQL weitere ganzzahlige Typen wie SMALLINT und BIGINT. Tabelle 5-4 zeigt eine Übersicht mit den entsprechenden Bitbreiten und Wertebereichen.

Tabelle 5-4: Bitbreiten und Wertebereiche der verschiedenen ganzzahligen SQL-Typen

Typ	Bit	Wertebereich	UNSIGNED-Wertebereich
`TINYINT`	8	–128 bis +127	0 bis 255
`SMALLINT`	16	–32.768 bis +32.767	0 bis 65.535
`MEDIUMINT`	24	–8.388.608 bis +8.388.607	0 bis 16.777.215
`INT` *oder* `INTEGER`	32	–2.147.483.648 bis +2.147.483.647	0 bis 4.294.967.295
`BIGINT`	64	–9.223.372.036.854.775.808 bis +9.223.372.036.854.775.807	0 bis 18.446.744.073.709.551.615

Hinter jedem `INT`-Typ können Sie in runden Klammern die gewünschte Darstellungsbreite, einen Wert zwischen 1 und 255, angeben. Der Kommandozeilenclient mysql stellt die Werte daraufhin rechtsbündig in Spalten mit der angegebenen Breite dar. Ein Beispiel:

```
seriennr INT(20)
```

Die Option `UNSIGNED` speichert, wie bereits erwähnt, nur vorzeichenlose Werte. Damit verdoppelt sich der Wertebereich für positive Zahlen, während negative wegfallen. Für Seriennummern, Lagerbestände oder ähnliche grundsätzlich positive Informationen ist dies praktisch. Eine weitere, nicht so häufig eingesetzte Option ist `ZEROFILL`: Sämtliche nicht besetzten Stellen bis zur Anzeigebreite werden mit Nullen aufgefüllt. Angenommen, Sie definieren eine Spalte als `INT(10) ZEROFILL` und weisen einem Feld dieser Spalte den Wert 123 zu. Der angezeigte Wert ist in diesem Fall 0000000123. Beachten Sie, dass `ZEROFILL` automatisch `UNSIGNED` bedingt.

Werden ganzzahlige Felder durch die weiter unten behandelten Einfügeabfragen (SQL-Anweisung `INSERT`) mit Werten belegt, ergeben sich folgende Besonderheiten:

- Wird ein Wert eingegeben, der für den zulässigen Wertebereich eines Felds zu groß oder zu klein ist, wird der größte beziehungsweise kleinste zulässige Wert eingetragen und eine Warnung ausgegeben. Zum Beispiel erhält ein Feld vom Typ `TINYINT` den Wert 127, wenn Sie versuchen, 129 oder gar 278 einzutragen. Ein `INT UNSIGNED`-Feld bekommt dagegen den Wert 0, wenn ein negativer Wert eingefügt wird.
- Bei der Eingabe von Fließkommawerten wird gerundet: Werte bis einschließlich 0.5 über einer ganzen Zahl werden auf die nächstkleinere Zahl abgerundet, Zahlen mit höherem Wert aufgerundet. 17.2, 17.49 und auch 17.5 werden beispielsweise zu 17, 17.51 oder 17.7 dagegen zu 18. Wenn Sie selbst Kontrolle über den Rundungsprozess ausüben möchten, können Sie die im nächsten Kapitel vorgestellten mathematischen Funktionen von MySQL verwenden.

Eine besondere Sorte ganzzahliger Werte erlaubt der in MySQL 5.0.3 eingeführte Datentyp `BIT(`*n*`)`: Der Wert von *n* kann zwischen 1 und 64 liegen und gibt an, dass die Spalte die angegebene Bitbreite aufweisen soll. Als Werte für solche Bitfelder

können Sie `b'Binärwert'` angeben, also beispielsweise `b'1011101'` (entspricht dem Dezimalwert 93). Der Datentyp `BIT` ist nützlich, wenn Sie platzsparend mehrere Ja/Nein-Eigenschaften speichern möchten; in der Programmierung werden solche binären Eigenschaftsfelder oft als *Flags* bezeichnet.

Fließkommatypen

Viele numerische Daten sind keine ganzen Zahlen – beispielsweise Messwerte, Geldbeträge oder Umrechnungsfaktoren. In der Mathematik muss man sich keine Gedanken darüber machen; es gibt abbrechende Dezimalbrüche wie 0.25, periodische wie 0.33333333… (das Ergebnis der Berechnung 1/3) und schließlich nicht-abbrechende, nicht-periodische wie die Kreiszahl π oder die Eulersche Zahl e. Im Gegensatz dazu können Computer nur abbrechende Dezimalbrüche speichern, da sie Speicherplätze mit festgelegter Bitbreite dafür verwenden. Alle nicht-abbrechenden Dezimalbrüche werden deshalb nur mit endlicher Genauigkeit gespeichert, das heißt an irgendeiner Stelle gerundet. Dazu wird das wissenschaftliche Format – also $x * 10^n$ – verwendet, so dass die Anzahl der Nachkommastellen variabel ist.[4] Daher werden diese Zahlen als *Fließkommazahlen* (englisch *Floating Point Numbers*, weil das Dezimaltrennzeichen in der üblichen englischen Schreibweise ein Punkt ist) bezeichnet.

MySQL kennt für Fließkommazahlen zwei Typen unterschiedlicher Genauigkeit: `FLOAT` und `DOUBLE`, auch `REAL` genannt. `FLOAT`-Werte belegen 32 Bit, `DOUBLE`-Werte 64. Auf Wunsch können hinter der Typbezeichnung zwei durch Komma getrennte Zahlen in Klammern angegeben werden. Die erste ist die bereits bei den ganzzahligen Typen erwähnte Anzeigebreite, die sich hier nur auf die Stellen vor dem Komma bezieht, die zweite ist die Anzahl der Nachkommastellen, die angezeigt werden sollen. Beispielsweise stellt `FLOAT(5,2)` Zahlen mit bis zu fünf Stellen vor dem Komma ordentlich untereinander dar und zeigt genau zwei Nachkommastellen an. Bei einer Begrenzung der Nachkommastellen wird mathematisch korrekt gerundet: In einem `FLOAT(5,2)`-Feld wird 25.3349 noch als 25.33 dargestellt, aber Werte ab 25.335 als 25.34.

Manche Zahlen besitzen eine von vornherein festgelegte Anzahl von Nachkommastellen – das bekannteste Beispiel sind natürlich Währungsbeträge. Für diesen Zweck definiert SQL neben den Fließkommazahlen auch sogenannte Festkommawerte; der entsprechende Datentyp heißt `DECIMAL` oder `NUMERIC`. In älteren MySQL-Versionen werden diese Werte intern als Strings codiert, so dass sie relativ viel Speicher benötigen und zudem recht langsam verarbeitet werden; seit Version 5.0.3 werden sie dagegen binär gespeichert. Die Verwendung von `DECIMAL` ist nur dann sinnvoll, wenn die Anzeigebreite und die Anzahl der Nachkommastellen im Format `DECIMAL(m,n)` angegeben werden. Zum Beispiel legt `DECIMAL(6,2)` eine Anzeigebreite von sechs Stellen vor dem Komma fest und definiert zwei Nachkommastellen.

---

4 Intern wird ein binäres Speicherformat, also $x * 2^n$, benutzt.

Auch Fließ- und Festkommatypen können mit der Option UNSIGNED versehen werden. Dadurch werden negative Werte unzulässig, aber anders als bei den ganzzahligen Datentypen wird dadurch der zulässige Wertebereich für positive Zahlen nicht erweitert.

Datum und Uhrzeit

Viele alltägliche Informationen sind mit bestimmten Zeitpunkten verknüpft. Im Fall des Reisebüros sind etwa Flugdatum und -uhrzeit besonders wichtig. MySQL stellt folgende verschiedene Spaltentypen zur Speicherung von Daten und Uhrzeiten zur Verfügung: DATETIME, DATE, TIME, YEAR und TIMESTAMP. DATETIME speichert einen vollständigen Zeitpunkt im Format JJJJ-MM-TT hh:mm:ss, also beispielsweise 2007-03-12 17:26:48. DATE enthält ein Datum ohne Uhrzeit, etwa 2007-03-19. In TIME-Feldern wird dagegen nur eine Uhrzeit abgelegt, zum Beispiel 17:28:32. YEAR-Felder enthalten ausschließlich eine Jahreszahl wie 2007.

TIMESTAMP-Felder erfüllen einen etwas anderen Zweck: Zum Zeitpunkt der Entstehung eines Datensatzes werden automatisch das aktuelle Datum und die aktuelle Uhrzeit eingefügt. Das Format von TIMESTAMP-Spalten entspricht DATETIME. Es ist allerdings ganz einfach, auch in alle anderen Arten von Datums- und Uhrzeitspalten den aktuellen Zeitpunkt einzufügen: Die MySQL-Funktion NOW() – ohne Argumente – liefert Systemdatum und -uhrzeit im genannten Format; der Datenbankserver sucht sich automatisch die für den jeweiligen Typ benötigten Bestandteile aus.

Daten und Uhrzeiten können bis zu einem gewissen Grad unvollständig eingegeben werden: Wenn Sie bei einem DATETIME-Feld die Uhrzeit weglassen, wird sie automatisch auf 00:00:00 gesetzt. Werden die Sekunden oder gar Minuten und Sekunden einer Uhrzeit ausgelassen, erhalten sie ebenfalls den Wert 0. Zum Beispiel wird 2007-03-14 in einer DATETIME-Spalte als 2007-03-14 00:00:00 abgelegt.

Eine Besonderheit ergibt sich bei der Eingabe zweistelliger Jahreszahlen: 00 bis 69 werden automatisch zu 2000 bis 2069, während 70 bis 99 in 1970 bis 1999 konvertiert werden. 07-03-14 steht also für 2007-03-14, 95-03-15 ist dagegen 1995-03-15.

Übrigens müssen Datums- und Uhrzeitwerte in MySQL (zum Beispiel bei der Eingabe per INSERT-Abfrage) in einfache oder doppelte Anführungszeichen gesetzt werden. Beispiele: "2007-03-18" oder '2007-03-21 17:56:32'. Die Alternative besteht darin, sie numerisch, aber ohne jegliche Trennzeichen im Format JJJJMMTThhmmss anzugeben, also beispielsweise 20070317132148 für den 17.03.2007, 13:21:48 Uhr.

Zeichenketten

Textinformationen kommen in einer Datenbank sehr häufig vor. In der Reisebüro-Datenbank müssen beispielsweise die Namen von Personen, Straßen, Städten und Hotels verwaltet werden. MySQL stellt zwei verschiedene Grundtypen zur Speiche-

rung kürzerer Textdaten (Zeichenketten oder Strings) zur Verfügung: CHAR und VARCHAR. Der Hauptunterschied: CHAR-Felder besitzen eine festgelegte Länge, die gegebenenfalls durch Leerzeichen aufgefüllt wird, während VARCHAR-Spalten eine variable Größe bis zur festgelegten Maximallänge besitzen (je nach festgelegter Höchstlänge werden ein bis zwei zusätzliche Zeichen zur Speicherung der aktuellen Länge benötigt). Aufgrund dieser Tatsache werden CHAR-Felder etwas schneller verarbeitet, VARCHAR verbraucht dagegen effektiv weniger Speicher.

Bei der Definition beider Typen wird die Angabe einer Länge in Zeichen erwartet. Beispiele: CHAR(10) oder VARCHAR(50). CHAR(10) besitzt eine feste Länge von zehn Zeichen, während in VARCHAR(50) maximal 50 Zeichen hineinpassen. CHAR-Felder können eine Länge zwischen 0 und 255 besitzen. Wenn Sie die Längenangabe weglassen, wird automatisch CHAR(1) eingestellt. Die maximale Länge von VARCHAR-Feldern konnte bisher ebenfalls 255 Zeichen betragen; seit MySQL 5.0.3 sind dagegen 65.535 Zeichen möglich.



Wenn statt Textdaten kurze Binärinformationen (nicht unbedingt Zahlen, sondern allgemeine binäre Datenblöcke) gespeichert werden sollen, stellt MySQL seit Version 4.1 die speziellen Datentypen BINARY und VARBINARY zur Verfügung. Die Eigenschaften entsprechen CHAR beziehungsweise VARCHAR. Die Maßeinheit für Längenangaben ist hier allerdings Bytes, statt Zeichen (die je nach Zeichensatz mitunter aus mehreren Bytes bestehen); zudem werden solche Felder in binärer Reihenfolge sortiert.

Text- und Binärblöcke

Für umfangreichere Text- und Binärdaten stehen mehrere TEXT- beziehungsweise BLOB-Typen (*BLOB* steht für *Binary Large OBject*) mit unterschiedlicher Kapazität zur Verfügung. Tabelle 5-5 zeigt eine Übersicht über diese Spaltentypen mit der jeweils zulässigen Höchstlänge. Bei den BLOB-Typen handelt es sich um die maximale Anzahl von Bytes, bei den TEXT-Typen um Zeichen.

Tabelle 5-5: Vergleich der verschiedenen Text- und Binärblock-Datentypen

TEXT-Typ	BLOB-Typ	maximale Länge
TINYTEXT	TINYBLOB	255
TEXT	BLOB	65.535
MEDIUMTEXT	MEDIUMBLOB	16.777.215
LONGTEXT	LONGBLOB	4.294.967.295

BLOB-Spalten sind durchaus zur Speicherung von Multimediadaten wie Bildern oder Audiomaterial geeignet. In den Beispielen in diesem Buch wird kein Gebrauch davon gemacht, da es relativ umständlich ist, Bilder, die in Webseiten eingebettet werden sollen, aus einer Datenbank auszulesen. Wann immer Bilddateien zum Ein-

satz kommen, wird in den Datenbanktabellen lediglich deren URL beziehungsweise Dateiname gespeichert.

Aufzählungstypen

Manche Tabellenspalten können nur wenige, festgelegte Werte annehmen. In solchen Fällen wird unnötig Speicherplatz vergeudet, wenn immer wieder die gleichen Textinformationen gespeichert werden. Zudem können so leicht Inkonsistenzen entstehen, etwa wenn Sie sich bei der Eingabe vertippen.

Die klassische relationale Lösung besteht darin, die gewünschten festen Werte in einer separaten Tabelle zu speichern und in der entsprechenden Spalte nur auf den jeweiligen Primärschlüssel zu verweisen. Hier ein Beispiel für die Bad-Ausstattung der Hotels:

```
+-------+-----------------+
| bd_nr | bd_beschreibung |
+-------+-----------------+
|     1 | ohne            |
|     2 | WC              |
|     3 | Dusche          |
|     4 | Bad             |
+-------+-----------------+
```

Die folgende Ansicht zeigt nur zwei Zeilen und nicht alle Spalten der Hotel-Tabelle, um zu demonstrieren, wie die Bad-Informationen durch einen Bezug auf die zusätzliche Tabelle gespeichert würden:

```
+-------+------------------------+----------+--------+
| ht_nr | ht_name                | ht_stadt | ht_bad |
+-------+------------------------+----------+--------+
|     2 | Hotel Colonia          |        1 |      4 |
|    23 | The New Maitland Hotel |       20 |      3 |
+-------+------------------------+----------+--------+
```

Eine Auswahlabfrage müsste eine komplexe Verknüpfung herstellen, um die Bad-Daten als Text auszulesen. Die obige Auswahl ließe sich wie folgt mit ausgeschriebenem Badezimmertyp darstellen, falls die Bad-Tabelle *rb_bad* und die Hotel-Tabelle *rb_hotels* heißt:

```
SELECT ht_nr, ht_name, ht_stadt, bd_beschreibung AS bad
FROM rb_hotels, rb_bad
WHERE ht_bad=bd_nr AND (ht_nr=2 OR ht_nr=23);
```

Genauere Erläuterungen zur Erstellung von Auswahlabfragen, dem wichtigsten Bestandteil von SQL, erhalten Sie im nächsten Kapitel. Das Ergebnis sähe jedenfalls so aus:

```
+-------+------------------------+----------+--------+
| ht_nr | ht_name                | ht_stadt | bad    |
+-------+------------------------+----------+--------+
|     2 | Hotel Colonia          |        1 | Bad    |
|    23 | The New Maitland Hotel |       20 | Dusche |
+-------+------------------------+----------+--------+
```

Für eine so triviale Information wie die Badezimmerausstattung, auf die nicht mehr weiter Bezug genommen wird, ist diese Lösung überdimensioniert. Glücklicherweise bietet MySQL eine »leichtgewichtige« Alternative in Form der sogenannten *Aufzählungstypen* `ENUM` und `SET`: Mit der Typdefinition wird eine festgelegte Menge von Textwerten gespeichert; die eigentlichen Einträge für eine solche Spalte sind numerische Verweise auf die jeweilige Elementnummer dieser Menge. Im Grunde geschieht hinter den Kulissen also dasselbe wie bei der gezeigten Tabellenverknüpfung. Für einfache, nicht mit weiteren Tabellen verknüpfte Spalten ist diese Lösung allerdings benutzerfreundlicher und auch etwas schneller.

Das Schema zur Definition von `ENUM`-Spalten sieht wie folgt aus:

```
Spaltenname ENUM('Wert1', 'Wert2', ...)
```

Das Badezimmerbeispiel besitzt also folgende Typdefinition:

```
ht_bad ENUM('ohne', 'WC', 'Dusche', 'Bad')
```

Intern werden die möglichen Einträge ab 1 durchnummeriert – statt `'Dusche'` könnte also beispielsweise 3 eingefügt werden. Angezeigt wird allerdings immer die Zeichenkette, auf die der gespeicherte Wert verweist. Der spezielle interne Wert 0 ist Fällen vorbehalten, in denen kein gültiger Inhalt ausgewählt wurde.

Der Spaltentyp `SET` geht einen Schritt weiter: Er erlaubt die Auswahl mehrerer Elemente der Menge. Intern werden die Werte als Bitfelder gespeichert. Dem ersten möglichen Eintrag wird das niedrigstwertige Bit (numerischer Wert 1) zugeordnet, dem nächsten das zweite Bit von rechts (Wert 2), dem nachfolgenden das dritte Bit (4), dem vierten 8 und so weiter. Hier ein Beispiel für mögliche Extras einer Hotelbuchung (wird im Praxisbeispiel dieses Buchs nicht verwendet):

```
ht_extras SET('Nichtraucher', 'Meerblick', 'Pool-Benutzung',
'Begrüßungscocktail', 'Vegetarier')
```

Der Eintrag `"Nichtraucher,Begrüßungscocktail"` besitzt den internen Wert 5 (binär 01001); `"Meerblick,Pool-Benutzung,Vegetarier"` wäre dem Wert 14 (10110) zugeordnet. Auch bei `SET`-Feldern wird in der Praxis stets der Textinhalt angezeigt, und zwar in Form einer durch Kommata getrennten Liste.

Schlüssel und Indizes

In Datenbanken muss oft nach einzelnen Informationen gesucht werden. Besonders in umfangreicheren Datensammlungen können solche Suchläufe allerdings ziemlich langwierig werden. Die Lösung besteht darin, wichtige Tabellenspalten mit einem *Index* zu versehen. Die Funktionsweise ähnelt dem alphabetischen Index am Ende dieses Buchs: Die in der Spalte vorkommenden Werte werden in sortierter Reihenfolge gespeichert und mit einer Referenz auf ihren tatsächlichen Speicherplatz in der Tabelle versehen. Auf diese Weise verringert sich die Suchdauer erheblich, und auch das Sortieren nach der entsprechenden Spalte wird beschleunigt.

MySQL definiert mehrere Arten von Indizes, die für verschiedene Einsatzzwecke geeignet sind:

- `INDEX` oder `KEY`: Standardindex, geeignet für alle Tabellen- und Spaltentypen.
- `UNIQUE`: Index mit der zusätzlichen Einschränkung, dass jeder Feldwert nur genau einmal in der Tabelle vorkommen darf.
- `FULLTEXT`: Optimierter Index für die Volltextsuche; ist nur für `CHAR`-, `VARCHAR`- und `*TEXT`-Spalten verfügbar, und zwar ausschließlich in MyISAM-Tabellen.
- `PRIMARY KEY`: Der Primärschlüssel ist mehr als ein einfacher Index. Wie bereits in den Abschnitten zum Tabellendesign besprochen, kennzeichnet der Primärschlüssel die Datensätze einer Tabelle eindeutig, damit aus einer anderen Tabelle heraus Bezug auf sie genommen werden kann. Das impliziert automatisch die Option `UNIQUE`.

Es gibt zwei verschiedene Methoden, um Indizes zu definieren: Entweder werden sie unmittelbar bei der Erstellung einer Tabelle eingefügt, oder sie werden nachträglich erzeugt.

In der Tabellendefinition werden die Indexoptionen nach den Spaltendefinitionen eingetragen:

```
CREATE TABLE Tabellenname (
   Feld1 Datentyp [Optionen],
   Feld2 Datentyp [Optionen],
   ...
   PRIMARY KEY (Feldname),
   INDEX (Feldname),
   UNIQUE (Feldname),
   ...
);
```

`PRIMARY KEY` kann auch direkt in der gewünschten Felddefinition stehen:

```
CREATE TABLE Tabellenname (
   Feld1 Datentyp [Optionen] PRIMARY KEY,
   Feld2 Datentyp [Optionen],
   ...
   PRIMARY KEY (Feldname),
   INDEX (Feldname),
   UNIQUE (Feldname),
   ...
);
```

Hier sehen Sie beide Versionen für die Kunden-Tabelle mit einem einfachen Index auf die Nachnamen:

```
CREATE TABLE rb_kunden (
   kd_nr INT AUTO_INCREMENT,
   kd_name VARCHAR(40),
   kd_vorname VARCHAR(40),
   kd_gdatum DATE,
```

```
   kd_bemerk VARCHAR(100),
   PRIMARY KEY (kd_nr),
   INDEX (kd_name)
);

CREATE TABLE rb_kunden (
   kd_nr INT AUTO_INCREMENT PRIMARY KEY,
   kd_name VARCHAR(40),
   kd_vorname VARCHAR(40),
   kd_gdatum DATE,
   kd_bemerk VARCHAR(100),
   INDEX (kd_name)
);
```

Optional können Sie den Klammern einen Indexnamen voranstellen. Die folgende Variante verwendet die Bezeichnung *name_key*:

```
CREATE TABLE rb_kunden (
   kd_nr INT AUTO_INCREMENT PRIMARY KEY,
   kd_name VARCHAR(40),
   kd_vorname VARCHAR(40),
   kd_gdatum DATE,
   kd_bemerk VARCHAR(100),
   INDEX name_key (kd_name)
);
```

Ein Index kann sich auch auf mehrere Spalten beziehen. In der Kunden-Tabelle könnte es beispielsweise sinnvoll sein, die Kombination aus Vor- und Nachname separat zu indizieren, da diese häufig verwendet wird. Die nachfolgende Version definiert einen entsprechenden zusätzlichen Schlüssel namens *full_name_key*:

```
CREATE TABLE rb_kunden (
   kd_nr INT AUTO_INCREMENT PRIMARY KEY,
   kd_name VARCHAR(40),
   kd_vorname VARCHAR(40),
   kd_gdatum DATE,
   kd_bemerk VARCHAR(100),
   INDEX name_key (kd_name),
   INDEX full_name_key (kd_vorname, kd_name)
);
```

Die zweite Variante – das nachträgliche Hinzufügen eines Index – geschieht entweder mithilfe einer Tabellenmanipulationsabfrage (`ALTER TABLE`) oder einer speziellen `CREATE INDEX`-Abfrage:

```
ALTER TABLE Tabellenname ADD [FULLTEXT|UNIQUE] INDEX [Indexname](Feldname1, ...);
```

beziehungsweise:

```
CREATE INDEX Indexname ON Tabellenname (Feldname1, ...);
```

Hier als Beispiel die `CREATE INDEX`-Variante für die Definition des zuletzt gezeigten *full_name_key*:

```
CREATE INDEX full_name_key ON rb_kunden (kd_vorname, kd_nachname);
```

Mithilfe von ALTER TABLE ließe sich dieselbe Abfrage wie folgt formulieren:

```
ALTER TABLE rb_kunden
ADD INDEX full_name_key (kd_vorname, kd_nachname);
```

Intern wird übrigens auch beim Aufruf von CREATE INDEX die zweite Variante aufgerufen.

Und schließlich ermöglicht ALTER TABLE es, einen Index wieder zu entfernen:

```
ALTER TABLE Tabellenname DROP INDEX Indexname;
```

Eine Abfrage zum Löschen des Index *full_name_key* sähe zum Beispiel so aus:

```
ALTER TABLE rb_kunden DROP INDEX full_name_key;
```

Falls kein separater Indexname definiert wurde, ist er mit dem Namen der (ersten) indizierten Spalte (in unserem Beispiel kd_vorname) identisch.

Im Zusammenhang mit Indizes, insbesondere mit Primärschlüsseln, wird häufig die Option AUTO_INCREMENT eingesetzt: Die Werte im entsprechenden Feld werden automatisch fortlaufend durchnummeriert. Dabei werden Werte aus gelöschten Datensätzen nicht wieder neu vergeben, um die Datenintegrität zu wahren. Pro Tabelle ist maximal eine AUTO_INCREMENT-Spalte zulässig; ihr Datentyp muss numerisch sein (zum Beispiel INT). Die Schreibweise einer solchen Definition sieht immer wie folgt aus:

```
Spaltenname Datentyp [Typoptionen] AUTO_INCREMENT [Indexart]
```

Das folgende Beispiel stammt nicht aus der Reisebüro-Tabelle. Es definiert eine automatisch durchnummerierte, auf zehn Stellen mit Nullen aufzufüllende Artikelnummer als Primärschlüssel:

```
artikelnr INT(10) ZEROFILL AUTO_INCREMENT PRIMARY KEY
```

Definierte Fremdschlüssel

Zu plausiblen Beziehungen zwischen verschiedenen Tabellen gehört mehr als die Einhaltung der weiter oben erläuterten Normalisierungsregeln. Wichtig ist nämlich auch, die Konsistenz der Beziehungen – die sogenannte *referenzielle Integrität* – im laufenden Betrieb zu überwachen. Beispielsweise muss ein Datensatz gelöscht werden, wenn er sich auf eine ebenfalls gelöschte Zeile einer anderen Tabelle bezieht. Stellen Sie sich im Reisebüro-Beispiel vor, eine Stadt würde komplett aus dem Programm genommen. Natürlich müssten dann auch sämtliche Hotels dieser Stadt aus der Hotel-Tabelle verschwinden. Andernfalls tritt eine sogenannte Lösch-Anomalie auf. Ein ähnliches Problem sind Update-Anomalien, bei denen die Änderung eines Schlüsselelements nicht auf abhängige Datensätze einer anderen Tabelle übertragen wird.

In MySQL-Datenbanken konnte man solche Sicherheitsvorkehrungen bis vor Kurzem nur durch eine entsprechende Absicherung der (externen) Datenbankanwendungen treffen. Beispielsweise sind ältere Versionen von phpMyAdmin damit

ausgestattet. Standard-SQL kennt dagegen schon länger sogenannte Foreign-Key-Constraints, die eine Spalte ausdrücklich als Fremdschlüssel kennzeichnen und so darauf achten, dass keine Anomalien aufreten können.

Seit MySQL 3.23.44 stehen diese Constraints für InnoDB-Tabellen zur Verfügung. Die Spalten, die miteinander verknüpft werden, müssen in beiden Tabellen einen Index besitzen. Bei mehrspaltigen Verknüpfungen müssen sie in beiden Tabellen hintereinander in derselben Reihenfolge vorliegen. Die grundlegende Syntax zum Erstellen eines Constraints sieht folgendermaßen aus:

```
[CONSTRAINT Name] FOREIGN KEY (Spalte1[, Spalte2 ...])
REFERENCES AndereTabelle (Spalte1[, Spalte2 ...])
```

Genau wie die diversen Indexklauseln kann auch dieses Konstrukt sowohl in eine `CREATE TABLE`-Anweisung als auch (mit vorangestelltem `ADD`) in eine `ALTER TABLE`-Anweisung eingesetzt werden.

In einer Webanwendung nützen `FOREIGN KEY`-Definitionen nicht besonders viel, weil jegliche Art von fehlerhaften Eingaben sowieso mit benutzerfreundlichen Meldungen abgefangen werden müssen. Deshalb hier ein separates Beispiel mit einer neuen, kleinen Datenbank:

Erstellen Sie eine neue Datenbank namens *foreign_test* und legen Sie sie als Standarddatenbank fest:

```
mysql> CREATE DATABASE foreign_test;
mysql> USE foreign_test;
```

Erstellen Sie eine zweispaltige Tabelle namens *laender*:

```
mysql> CREATE TABLE laender (
    ->   la_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   la_name VARCHAR(40)
    -> )
    -> ENGINE=InnoDB;
```

Fügen Sie folgende Datensätze ein:

```
mysql> INSERT INTO laender (la_name) VALUES
    -> ('Deutschland'),
    -> ('Frankreich'),
    -> ('Italien');
```

Erstellen Sie eine weitere Tabelle namens *staedte* – sie enthält den Constraint auf die Nummer des entsprechenden Landes:

```
mysql> CREATE TABLE staedte (
    ->   st_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   st_name VARCHAR (40),
    ->   st_land INT,
    ->   INDEX (st_land),
    ->   FOREIGN KEY (st_land) REFERENCES laender(la_id)
    -> )
    -> ENGINE=InnoDB;
```

Auch in diese Tabelle werden nun Werte eingefügt:

```
mysql> INSERT INTO staedte (st_name, st_land) VALUES
    -> ('Köln', 1),
    -> ('Bonn', 1),
    -> ('Paris', 2),
    -> ('Lyon', 2),
    -> ('Rom', 3),
    -> ('Neapel', 3);
```

Lassen Sie sich zur Kontrolle eine `JOIN`-Ansicht der Städte- und Ländernamen anzeigen:

```
mysql> SELECT la_nr, st_nr
    -> FROM staedte INNER JOIN laender ON la_id=st_land;
+-------------+---------+
| la_name     | st_name |
+-------------+---------+
| Deutschland | Köln    |
| Deutschland | Bonn    |
| Frankreich  | Paris   |
| Frankreich  | Lyon    |
| Italien     | Rom     |
| Italien     | Neapel  |
+-------------+---------+
```

Nun wird versucht, Frankreich aus der Länderliste zu löschen:

```
mysql> DELETE FROM laender WHERE la_id=2;
ERROR 1217 (23000): Cannot delete or update a parent row:
 a foreign key constraint fails
```

Wie Sie sehen, weigert sich der MySQL-Server mit einer Fehlermeldung – weil Städte vorhanden sind, die in Frankreich liegen, so dass dieser Datensatz benötigt wird. Es gibt Constraint-Optionen, die das Löschen oder Ändern automatisch auf die abhängigen Datensätze übertragen. Deshalb erfahren Sie hier zunächst, wie ein Constraint gelöscht wird; anschließend wird er mit diesen Optionen wieder hinzugefügt. Erstes Problem: Wenn Sie keine `CONSTRAINT`-*Name*-Klausel gesetzt haben, besitzt der Constraint einen automatisch von MySQL vergebenen Namen. Dieser lässt sich aber über eine `SHOW CREATE TABLE`-Abfrage ermitteln:

```
mysql> SHOW CREATE TABLE staedte \G
*************************** 1. row ***************************
       Table: staedte
Create Table: CREATE TABLE `staedte` (
  `st_id` int(11) NOT NULL auto_increment,
  `st_name` varchar(40) default NULL,
  `st_land` int(11) default NULL,
  PRIMARY KEY (`st_id`),
  KEY `st_land` (`st_land`),
  CONSTRAINT `staedte_ibfk_1` FOREIGN KEY (`st_land`) REFERENCES `laender`
  (`la_id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Der Constraint heißt also *staedte_ibfk_1*. Nachdem der Name bekannt ist, kann er gelöscht werden:

```
mysql> ALTER TABLE staedte
    -> DROP FOREIGN KEY staedte_ibfk_1;
```

Als Nächstes wird er mithilfe einer weiteren `ALTER TABLE`-Abfrage neu erstellt – diesmal mit festgelegtem Namen:

```
mysql> ALTER TABLE staedte
    -> ADD CONSTRAINT land
    -> FOREIGN KEY (st_land) REFERENCES laender (la_id)
    -> ON DELETE CASCADE ON UPDATE CASCADE;
```

Die hinzugefügten Optionen `ON DELETE CASCADE` und `ON UPDATE CASCADE` bedeuten, dass Lösch- beziehungsweise Änderungsabfragen an die abhängige Tabelle weitergegeben werden.

Nun kann Frankreich aus der Tabelle *laender* entfernt werden:

```
mysql> DELETE FROM laender WHERE la_id=2;
Query OK, 1 row affected (0.06 sec)
```

Damit sind auch alle französischen Städte automatisch verschwunden, wie die Wiederholung der SELECT-Abfrage zeigt:

```
mysql> SELECT la_name, st_name
    ->  FROM staedte INNER JOIN laender ON la_id=st_land;
+-------------+---------+
| la_name     | st_name |
+-------------+---------+
| Deutschland | Köln    |
| Deutschland | Bonn    |
| Italien     | Rom     |
| Italien     | Neapel  |
+-------------+---------+
```

Auch Änderungen in der Spalte *la_id* werden auf *st_land* übertragen. Hier als Beispiel die relativ sinnlose Änderung der ID von Italien auf den (nun frei gewordenen) Wert 2:

```
mysql> UPDATE laender SET la_id=2 WHERE la_id=3;
```

Ein SELECT der gewünschten Spalten in *staedte* zeigt, dass die Nummer auch hier angepasst wurde:

```
mysql> SELECT st_name, st_land FROM staedte;
+---------+---------+
| st_name | st_land |
+---------+---------+
| Köln    |       1 |
| Bonn    |       1 |
| Rom     |       2 |
| Neapel  |       2 |
+---------+---------+
```

Daten einfügen

Nachdem die Datenbanken und Tabellen fertiggestellt sind, müssen Daten eingefügt werden. Dazu wird die SQL-Anweisung INSERT verwendet; Beispiele wurden bereits in früheren Kapiteln gezeigt. Da die Datenbank *reisebuero* bereits »ab Werk« alle vorgegebenen Daten enthält, kommt für die Beispiele in diesem Abschnitt zur Abwechslung wieder einmal das *gewinnspiel* aus den vorangehenden Kapiteln zum Einsatz. Zunächst müssen Sie diese Datenbank als Standard einstellen:

```
mysql> USE gewinnspiel;
```

Die klassische Syntax von INSERT sieht folgendermaßen aus:

```
INSERT INTO Tabelle [(Feld1, Feld2, ...)]
VALUES (Wert1, Wert2, ...)
[, (Wert1, Wert2, ...), ...];
```

Eine Liste von Feldnamen brauchen Sie nur anzugeben, wenn nicht für jede Spalte ein Wert eingegeben werden soll.

Das folgende Beispiel trägt einen neuen Benutzer in die Tabelle *gw_teilnehmer* ein, wobei alle Spalten außer dem per AUTO_INCREMENT ausgefüllten Primärschlüssel explizit benannt werden:

```
mysql> INSERT INTO gw_teilnehmer (tn_uname, tn_email, tn_interest)
    -> VALUES ('Heinz', 'heinz@example.net', 1);
Query OK, 1 row affected (0.03 sec)
```

Die Meldung Query OK zeigt bereits, dass das Einfügen erfolgreich war. Eine einfache SELECT-Abfrage bringt Gewissheit:

```
mysql> SELECT * FROM gw_teilnehmer;
+-------+----------+-------------------+-------------+
| tn_id | tn_uname | tn_email          | tn_interest |
+-------+----------+-------------------+-------------+
|     1 | Klaus    | klaus@example.com | 2           |
|     2 | Sabine   | sabine@example.net | 3          |
|     3 | Hans     | hans@example.org  | 4           |
|     4 | Heinz    | heinz@example.net | 1           |
+-------+----------+-------------------+-------------+
```

Wenn Sie trotz Anwesenheit von AUTO_INCREMENT- oder TIMESTAMP-Spalten auf die Angabe der Felder verzichten möchten, können Sie dieselben einfach mit NULL füllen – die korrekten Werte werden automatisch eingetragen. Hier die entsprechende Vorgehensweise für *gw_teilnehmer*:

```
mysql> INSERT INTO gw_teilnehmer
    -> VALUES (NULL, "Herbert", "herbert@gmz.de", 3);
Query OK, 1 row affected (0.02 sec)
```

Sie können auch mehrere Datensätze gleichzeitig einfügen, indem Sie beliebig viele von ihnen durch Kommata trennen. Hier sind beispielsweise noch zwei weitere Gewinnspielteilnehmerinnen:

```
mysql> INSERT INTO gw_teilnehmer (tn_uname, tn_email, tn_interest)
    -> VALUES
    -> ("Jana", "jana@aof.com", 2),
    -> ("Angelika", "angelika@coldmail.com", 4);
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Sie können sich auch aus anderen Tabellen bedienen, um Daten einzufügen. Dazu wird eine INSERT-Abfrage mit einem SELECT gekoppelt. Wichtig ist, dass die Anzahl und Datentypen der ausgewählten Spalten mit dem Ziel übereinstimmen. Das folgende Beispiel »entführt« die ersten drei Kunden aus der Kundentabelle des Reisebüros in die Tabelle mit den Gewinnspielteilnehmern:

```
mysql> INSERT INTO gw_teilnehmer (tn_uname, tn_email, tn_interest)
    -> SELECT kd_vorname, kk_mail, 2
    -> FROM reisebuero.rb_kunden INNER JOIN
    -> reisebuero.rb_kundenkontakte
    -> ON kd_nr=kk_nr AND kd_nr <= 3;
Query OK, 3 rows affected (0.41 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Die etwas umständliche Angabe der Datenbank *reisebuero* ist hier erforderlich, weil sie zurzeit nicht die Standarddatenbank ist. Als bevorzugte Stadt wird die Konstante 2 eingefügt, da die Quelltabellen keine adäquate Information enthalten. Die komplexe INNER JOIN-Syntax, die die Tabellen *rb_kunden* und *rb_kundenkontakte* anhand der identischen Kundennummern miteinander verknüpft, wird im nächsten Kapitel genau erläutert.

Ein abschließendes SELECT zeigt das Ergebnis der letzten drei Einfügeabfragen:

```
mysql> SELECT * FROM gw_teilnehmer;
+-------+----------+-----------------------+-------------+
| tn_id | tn_uname | tn_email              | tn_interest |
+-------+----------+-----------------------+-------------+
|     1 | Klaus    | klaus@example.com     | 2           |
|     2 | Sabine   | sabine@example.net    | 3           |
|     3 | Hans     | hans@example.org      | 4           |
|     4 | Heinz    | heinz@example.net     | 1           |
|     5 | Herbert  | herbert@gmz.de        | 3           |
|     6 | Jana     | jana@aof.com          | 2           |
|     7 | Angelika | angelika@coldmail.com | 4           |
|     8 | Ludwig   | becker@berlinerland.de | 2          |
|     9 | Heinrich | meier@aol-lokal.de    | 2           |
|    10 | Klaus    | schmitz@kajs.com      | 2           |
+-------+----------+-----------------------+-------------+
```

In diesem Kapitel:

- Auswahlabfragen
- SQL-Ausdrücke und -Funktionen
- Weitere Abfragetypen

KAPITEL 6
SQL-Abfragen

Denn dies wird ja als der erste Schlüssel zur Weisheit bestimmt: das beständige und häufige Fragen.
Pierre Abélard

In den bisherigen Kapiteln haben Sie bereits einzelne SQL-Abfragen kennengelernt. Nun wird es Zeit, dieses Thema zu vertiefen und systematisch zu behandeln. Im vorliegenden Kapitel werden vor allem die grundlegenden SQL-Techniken für den »Datenbankalltag« vorgestellt; sie genügen, um im übernächsten Kapitel die unterschiedlichsten Webanwendungen zu erstellen. Das nächste Kapitel stellt dagegen etwas knapper einige weiterführende SQL-Features vor, die in Webanwendungen in der Regel nicht benötigt werden.

Auswahlabfragen

Die wichtigste Form der SQL-Abfrage ist die *Auswahlabfrage*. Sie verwendet die SQL-Anweisung `SELECT`, um Informationen aus Datenbanktabellen oder auch vom Datenbankserver selbst zu erhalten und auszugeben.

Die einfachste Form einer `SELECT`-Abfrage wählt alle Datensätze einer Tabelle aus:

```
SELECT * FROM Tabellenname;
```

Das folgende Beispiel zeigt alle Datensätze der Tabelle *rb_kunden* an:

```
mysql> SELECT * FROM rb_kunden;
+-------+---------+------------+------------+-----------+
| kd_nr | kd_name | kd_vorname | kd_gdatum  | kd_bemerk |
+-------+---------+------------+------------+-----------+
|     1 | Schmitz | Klaus      | 1967-08-03 |           |
|     2 | Meier   | Heinrich   | 1976-09-24 |           |
|     3 | Becker  | Ludwig     | 1955-11-27 |           |
|     4 | Huber   | Michael    | 1965-08-11 |           |
|     5 | Jäger   | Siegfried  | 1952-01-17 |           |
```

```
|      6 | Klein   | Klaus      | 1980-04-13 |           |
|      7 | Becker  | Michaela   | 1963-09-19 |           |
|      8 | Schmitz | Susanne    | 1970-02-07 |           |
|      9 | Vogel   | Martina    | 1975-06-14 |           |
|     10 | Gruber  | Annette    | 1954-03-18 |           |
|     11 | Berger  | Sven       | 1967-07-02 |           |
+-------+---------+------------+------------+-----------+
11 rows in set (0.06 sec)
```



Neben der Hauptaufgabe, Datensätze aus Tabellen auszuwählen, kann SELECT auch beliebige Ausdrücke auswerten und das Ergebnis anzeigen. Das können Sie beispielsweise für Berechnungen nutzen:

```
mysql> SELECT 7 * 6;
+-------+
| 7 * 6 |
+-------+
|    42 |
+-------+
```

Auch die Werte globaler Funktionen können Sie auf diese Weise abfragen – hier beispielsweise mehrere Spalten mit Systemdatum und -uhrzeit sowohl mit der MySQL-Version als auch mit der aktuellen Standarddatenbank:

```
mysql> SELECT NOW(), VERSION(), DATABASE();
+--------------------+-------------+------------+
| NOW()              | VERSION()   | DATABASE()  |
+--------------------+-------------+------------+
| 2007-05-01 19:46:48 | 5.1.18-beta- | gewinnspiel |
|                    | community-nt |            |
+--------------------+-------------+------------+
```

Wenn Sie nur einzelne Spalten einer Tabelle auslesen möchten, müssen Sie statt eines * die durch Kommata getrennten Feldnamen angeben:

```
SELECT Feld1[, Feld2, ...] FROM Tabellenname;
```

Hier ein Beispiel, das nur die Namen und Kürzel der Fluggesellschaften anzeigt:

```
mysql> SELECT ai_name, ai_kuerzel FROM rb_airlines;
+-----------------------+-----------+
| ai_name               | ai_kuerzel |
+-----------------------+-----------+
| Air France            | AF         |
| Austrian Airlines     | OS         |
| British Airways       | BA         |
| Germanwings           | 4U         |
| Lufthansa             | LH         |
| KLM                   | KM         |
| Iberia                | IB         |
| Alitalia              | AZ         |
```

```
| HLX                    | X3         |
| Olympic Airways        | OA         |
| Turkish Airlines (THY) | TK         |
+------------------------+------------+
```

Wenn Ihnen die Spaltenbeschriftungen durch die Spaltennamen nicht gefallen, können Sie die Klausel AS verwenden, um Ihre Ergebnisse beliebig umzubenennen. Dabei muss der Aliasname in Anführungszeichen stehen, falls er Leerzeichen enthält. Das folgende Beispiel zeigt die Namen, Einzel- und Doppelzimmerpreise der ersten fünf Hotels mit lesefreundlicheren Spaltentiteln an:

```
mysql> SELECT ht_name AS Hotel, ht_ezpreis AS Einzelzimmerpreis,
    -> ht_dzpreis AS Doppelzimmerpreis FROM rb_hotels LIMIT 0,5;
+------------------+-------------------+-------------------+
| Hotel            | Einzelzimmerpreis | Doppelzimmerpreis |
+------------------+-------------------+-------------------+
| Bergerhof        |                70 |               120 |
| Hotel Colonia    |               105 |               195 |
| Hotel de la Gare |                85 |               150 |
| Hotel au Jardin  |               110 |               200 |
| Otel Bahar       |                50 |                80 |
+------------------+-------------------+-------------------+
```

Das Schlüsselwort AS selbst kann sogar wegfallen. Beispiel:

```
mysql> SELECT ht_name Hotel, ht_ezpreis Einzelzimmer
    -> FROM rb_hotels LIMIT 0,3;
+------------------+--------------+
| Hotel            | Einzelzimmer |
+------------------+--------------+
| Bergerhof        |           70 |
| Hotel Colonia    |          105 |
| Hotel de la Gare |           85 |
+------------------+--------------+
```

Übrigens können Sie nicht nur Spalten oder berechneten Ergebnissen einen Aliasnamen zuweisen, sondern auch Tabellen. Dies ist nützlich, wenn Sie zwei verschiedene Werte aus derselben Tabelle benötigen, was weiter unten besprochen wird. Das folgende Beispiel weist der Tabelle *rb_kunden* jedenfalls den Ersatznamen *kunden* zu:

```
mysql> SELECT kd_name, kd_vorname, kd_mail FROM rb_kunden kunden
    -> LIMIT 0,4;
+---------+------------+------------------------+
| kd_name | kd_vorname | kd_mail                |
+---------+------------+------------------------+
| Schmitz | Klaus      | schmitz@kajs.com       |
| Meier   | Heinrich   | meier@aol-lokal.de     |
| Becker  | Ludwig     | becker@berlinerland.de |
| Huber   | Michael    | mhuber@d-online.de     |
+---------+------------+------------------------+
```

Die LIMIT-Klausel beschränkt die Anzahl der Abfrageergebnisse: Die erste Zahl gibt das Startelement (entsprechend der aktuellen Sortierung) ab 0 an, die zweite ist die maximale Anzahl gewünschter Datensätze.

Übrigens gibt eine Auswahlabfrage standardmäßig auch dann alle Datensätze aus, wenn mehrere identisch sind. Dies können Sie mithilfe der Klausel DISTINCT oder DISTINCTROW (Synonym) ändern. Das folgende Beispiel wählt den Badezimmertyp der ersten fünf Hotels aus:

```
mysql> SELECT ht_bad FROM rb_hotels LIMIT 0,5;
+--------+
| ht_bad |
+--------+
| Dusche |
| Bad    |
| Dusche |
| Bad    |
| Dusche |
+--------+
```

Möchten Sie dagegen herausfinden, welche der definierten Badtypen überhaupt in allen verfügbaren Hotels eingesetzt werden, können Sie die Abfrage wie folgt um ein DISTINCTROW ergänzen:

```
mysql> SELECT DISTINCTROW ht_bad FROM rb_hotels;
+--------+
| ht_bad |
+--------+
| Dusche |
| Bad    |
+--------+
```

Suchkriterien – die WHERE-Klausel

In der Regel werden nicht alle Datensätze einer Tabelle benötigt, sondern es soll eine Auswahl nach bestimmten Kriterien getroffen werden. Dafür ist die SQL-Klausel WHERE zuständig. Sie wird hinter FROM *Tabellenname* angegeben:

```
SELECT Feld[er]|* FROM Tabellenname WHERE Ausdruck;
```

Diese Abfrage wählt diejenigen Datensätze aus, auf die der verwendete Ausdruck zutrifft. Es gibt unzählige Möglichkeiten für die Formulierung dieses Ausdrucks. Am häufigsten wird eine Spalte mit einem bestimmten Wert verglichen. Das folgende Beispiel wählt nur die Fluggesellschaft mit dem Kürzel "4U" (Germanwings) aus:

```
mysql> SELECT ai_name, ai_kuerzel FROM rb_airlines
    -> WHERE ai_kuerzel="4U";
+-----------------------+------------+
| ai_name               | ai_kuerzel |
+-----------------------+------------+
| Germanwings           | 4U         |
+-----------------------+------------+
```

Vergleichsoperatoren

Tabelle 6-1 zeigt eine Übersicht der SQL-Vergleichsoperatoren. In der Spalte »Bedeutung« stehen *Op1* und *Op2* für die beiden Operanden (zu vergleichenden Werte), und zwar von links nach rechts gelesen.

Tabelle 6-1: Die MySQL-Vergleichsoperatoren

Operator	Name	Bedeutung
=	gleich	wahr, wenn *Op1* und *Op2* identisch sind
!= oder <>	ungleich	wahr, wenn *Op1* und *Op2* verschieden sind
<	kleiner als	wahr, wenn *Op1* einen niedrigeren Wert als *Op2* hat
>	größer als	wahr, wenn *Op1* einen höheren Wert als *Op2* hat
<=	kleiner oder gleich	wahr, wenn *Op1* einen niedrigeren oder denselben Wert wie *Op2* hat
>=	größer oder gleich	wahr, wenn *Op1* einen höheren oder denselben Wert wie *Op2* hat
<=>	NULL-sicheres Gleich	wie =, behandelt aber auch `NULL`-Werte korrekt

Für numerische Werte dürfte die Funktionsweise der Vergleichsoperatoren offensichtlich sein: Wie Sie aus der Mathematik wissen, gelten die Ausdrücke `1 != 4`, `-1 < 1` und `2 >= 2` als wahr. Dagegen sind `3 = 4`, `4 < 4` oder `3 >= 7` falsche Aussagen. Wenn Sie Vergleichsoperationen in einer `WHERE`-Klausel einsetzen, wählt MySQL nur diejenigen Datensätze aus, für die der Vergleich eine wahre Aussage ergibt. Das folgende Beispiel wählt die Namen und Kürzel der ersten vier Fluggesellschaften aus:

```
mysql> SELECT ai_name, ai_kuerzel FROM rb_airlines
    -> WHERE ai_nr <= 4;
+------------------------+------------+
| ai_name                | ai_kuerzel |
+------------------------+------------+
| Air France             | AF         |
| Austrian Airlines      | OS         |
| British Airways        | BA         |
| Germanwings            | 4U         |
+------------------------+------------+
```

Statt `ai_nr <= 4` hätten Sie in diesem Fall auch `ai_nr < 5` schreiben können; bei ganzzahligen Werten macht das keinen Unterschied. Natürlich gilt das nicht für Fließkommawerte.



Wie Sie sehen, muss die Spalte, auf der die `WHERE`-Klausel basiert, keineswegs mit ausgewählt werden. Im vorliegenden Beispiel verwendet der Vergleich den Primärschlüssel *ai_nr*, es werden aber nur *ai_name* und *ai_kuerzel* angezeigt.

Die Vergleichsoperationen können auch für String-Werte aller Art eingesetzt werden. Dabei wird die Position der einzelnen Zeichen im jeweiligen Zeichensatz als Vergleichskriterium eingesetzt. Anders als die meisten Programmiersprachen unter-

scheidet SQL standardmäßig nicht zwischen Groß- und Kleinschreibung, was in den meisten Fällen sehr praktisch ist. Allgemein lässt sich sagen, dass ein String »kleiner« als ein anderer ist, wenn er im Alphabet vor diesem an der Reihe ist. Zur Unterscheidung wird das erste unterschiedliche Zeichen verwendet. Zum Beispiel gilt `"London" < "Paris"` und `"Barcelona" < "Berlin"`.

In Einzelfällen bestimmt die Kollation, also die festgelegte Sortierreihenfolge, die Ergebnisse solcher Operationen. Wie bereits erwähnt, gibt es für Deutsch zwei mögliche Kollationen: *latin1_german1_ci* sortiert nach Wörterbuch, so dass die Umlaute *ä*, *ö* und *ü* mit den Buchstaben *a*, *o* und *u* gleichgesetzt werden. *latin1_german2_ci* ist dagegen die Telefonbuch-Kollation, bei der *ä*, *ö* und *ü* wie *ae*, *oe* beziehungsweise *ue* behandelt werden.

MySQL enthält weit über 100 verschiedene Kollationen. Ihr Name setzt sich aus mehreren, durch Unterstriche getrennten Komponenten zusammen: dem Zeichensatz (zum Beispiel *latin1* oder *utf8*), der Sprache (etwa *swedish* oder *turkish*) mit eventueller Nummerierung von Variationen (*german1* und *german2*) und der Sortiermethode (*ci*, *cs* oder *bin*). Während *ci* (case-insensitive) nicht zwischen Groß- und Kleinschreibung unterscheidet, kommt in *cs* (case-sensitive) jeder Großbuchstabe vor dem entsprechenden Kleinbuchstaben an die Reihe. *bin* richtet sich dagegen strikt nach der Reihenfolge der Zeichensätze selbst, so dass *alle* Großbuchstaben vor *allen* Kleinbuchstaben eingeordnet werden. Eine konkrete Liste der Kollationen erhalten Sie mithilfe der folgenden Anweisung:

```
SHOW COLLATION;
```

Da die Liste zu lang für ein Terminal- oder Eingabeaufforderungsfenster ist, sollten Sie zuvor eine Logdatei aktivieren (siehe Kapitel 4) oder gleich den Link *Zeichensätze und Kollationen* auf der phpMyAdmin-Startseite verwenden. Dieser enthält bequemerweise eine kurze Beschreibung jeder Kollation.

In der Datenbank *reisebuero* verwenden alle Tabellen konsequent die Kollation *latin1_german1_ci*, so dass innerhalb dieser Datenbank kein Raum für entsprechende Experimente existiert. Deshalb folgt hier ein kurzes separates Beispiel zum Ausprobieren. Zunächst sollten Sie eine separate Testdatenbank erstellen und als Standarddatenbank auswählen:

```
mysql> CREATE DATABASE texttest;
mysql> USE texttest;
```

In der neuen Datenbank wird zunächst eine Tabelle mit einer einzigen Spalte und der Kollation *latin1_german1_ci* erstellt:

```
mysql> CREATE TABLE deutsch (txt VARCHAR(50))
    -> COLLATE latin1_german1_ci;
```

In diese Tabelle sollen die vier Wörter »Bach«, »Buch«, »angeln« und »backen« eingetragen werden:

```
mysql> INSERT INTO deutsch VALUES
    -> ('Bach'),
    -> ('Buch'),
    -> ('angeln'),
    -> ('backen');
```

Nun werden zwei weitere Tabellen mit anderen Kollationen als Kopien von *deutsch* erstellt:

```
mysql> CREATE TABLE grossklein
    -> COLLATE latin1_general_cs
    -> SELECT * FROM deutsch;

mysql> CREATE TABLE binaer
    -> COLLATE latin1_bin
    -> SELECT * FROM deutsch;
```

Wenn Sie sich nun die Inhalte aller drei Tabellen sortiert anzeigen lassen, erkennen Sie die Unterschiede zwischen den Kollationen:

```
mysql> SELECT * FROM deutsch ORDER BY txt ASC;
+--------+
| txt    |
+--------+
| angeln |
| Bach   |
| backen |
| Buch   |
+--------+

mysql> SELECT * FROM grossklein ORDER BY txt ASC;
+--------+
| txt    |
+--------+
| angeln |
| Bach   |
| Buch   |
| backen |
+--------+

mysql> SELECT * FROM binaer ORDER BY txt ASC;
+--------+
| txt    |
+--------+
| Bach   |
| Buch   |
| angeln |
| backen |
+--------+
```

Die zum Sortieren verwendete Klausel `ORDER BY` wurde bereits in früheren Kapiteln erwähnt; im übernächsten Abschnitt wird sie ausführlich erläutert.

Sehen Sie nun noch ein weiteres Beispiel aus der Reisebüro-Datenbank: Es wählt die Namen und Vornamen aller Kunden aus, deren Namen mit *A* bis *M* beginnen, also *kleiner als »N«* sind, und sortiert sie nach den Nachnamen:

```
mysql> SELECT kd_vorname, kd_name FROM rb_kunden
    -> WHERE kd_name < "N" ORDER BY kd_name ASC;
+------------+---------+
| kd_vorname | kd_name |
+------------+---------+
| Ludwig     | Becker  |
| Michaela   | Becker  |
| Sven       | Berger  |
| Annette    | Gruber  |
| Michael    | Huber   |
| Siegfried  | Jäger   |
| Klaus      | Klein   |
| Heinrich   | Meier   |
+------------+---------+
```

Übrigens können Sie natürlich nicht nur Felder mit festen Ausdrücken vergleichen, sondern auch beliebige Felder miteinander. Die nachfolgende SQL-Anweisung wählt alle Städte aus, bei denen sich deutscher Name und Eigenname unterscheiden:

```
mysql> SELECT st_name, st_eigen FROM rb_staedte
    -> WHERE st_name != st_eigen;
+----------+-----------+
| st_name  | st_eigen  |
+----------+-----------+
| Rom      | Roma      |
| Athen    | Athina    |
| Brüssel  | Bruxelles |
| Warschau | Warszawa  |
| Prag     | Praha     |
| Venedig  | Venezia   |
| Lissabon | Lisboa    |
+----------+-----------+
```

Datums- und Uhrzeit-Spalten können ebenfalls mithilfe der Vergleichsoperatoren überprüft werden. Ein Zeitpunkt ist »kleiner als« ein anderer, wenn er vor diesem stattfindet. Genau wie bei der Eingabe können Sie auch bei Vergleichen einzelne Teile von Daten und Uhrzeiten eingeben. Das folgende Beispiel wählt alle Flüge aus, die ab September 2007 stattfinden:

```
SELECT * FROM rb_fluege WHERE fl_datum > "2007-09";
```



Vergleiche mit Datums- und Uhrzeitbestandteilen sind nur mit den verschiedenen Arten von Ungleichheitsoperatoren möglich; für = sind sie sinnlos und liefern daher ein leeres Ergebnis: Ein Zeitpunkt kann dem Kriterium > "2007-09" genügen, wenn er am 2007-09-01 oder später stattfindet; es gibt aber keinen Zeitpunkt, der *genau* "2007-09" entspricht.

Der spezielle Operator <=> wurde in MySQL 3.23 eingeführt. Er korrigiert das Fehlverhalten von = bei NULL-Werten, das heißt bei Feldern ohne Inhalt. Beispielsweise ergibt der Ausdruck *Feldname* = NULL für Felder mit dem Inhalt NULL nicht etwa TRUE, sondern NULL. SELECT liefert also nicht die entsprechenden Datensätze, sondern ein leeres Ergebnis. *Feldname* <=> NULL gibt dagegen alle Datensätze zurück, in denen die Spalte *Feldname* leer ist. Dasselbe gilt für Vergleiche zwischen anderen Werten, wenn sie NULL sind. Eine Alternative sind die speziellen Operationen IS NULL beziehungsweise IS NOT NULL, die Sie mithilfe der unten besprochenen AND- oder OR-Verknüpfungen mit anderen Bedingungen kombinieren können.

Mustervergleiche mit LIKE

Für Zeichenketten sind die Vergleichsoperatoren manchmal nicht ausreichend: In vielen Fällen muss nach Textmustern gesucht werden, etwa um Strings zu finden, die eine bestimmte Zeichenkombination enthalten. Die traditionelle SQL-Methode für die Mustersuche ist der Operator LIKE, der zwei einfache Platzhalter zur Verfügung stellt. Moderner und erheblich leistungsfähiger sind die über den Operator REGEXP bereitgestellten regulären Ausdrücke (siehe nächster Abschnitt).

In einem Konstrukt mit dem Schema *Feldname* LIKE *Muster* stehen die beiden folgenden Platzhalter zur Verfügung: Das Prozentzeichen (%) steht für beliebig viele beliebige Zeichen, während der Unterstrich (_) genau ein beliebiges Zeichen repräsentiert. Das folgende Beispiel wählt die vollständigen Namen aller Kunden aus, deren Vornamen mit *M* beginnen:

```
mysql> SELECT kd_vorname, kd_name FROM rb_kunden
    -> WHERE kd_vorname LIKE "M%";
+------------+---------+
| kd_vorname | kd_name |
+------------+---------+
| Martina    | Vogel   |
| Michael    | Huber   |
| Michaela   | Becker  |
+------------+---------+
```

Das nächste Beispiel wählt die Namen und Kürzel aller Fluggesellschaften aus, deren Namen mindestens ein *n* enthalten:

```
mysql> SELECT ai_name, ai_kuerzel FROM rb_airlines
    -> WHERE ai_name LIKE "%n%";
```

```
+-----------------------+------------+
| ai_name               | ai_kuerzel |
+-----------------------+------------+
| Air France            | AF         |
| Austrian Airlines     | OS         |
| Germanwings           | 4U         |
| Lufthansa             | LH         |
| Turkish Airlines (THY)| TK         |
+-----------------------+------------+
```

Mit der nachfolgenden Abfrage werden alle Flughäfen ausgewählt, deren Websites mit der Top-Level-Domain *.de* (Deutschland) enden:

```
mysql> SELECT ap_kuerzel, ap_url FROM rb_airports
    -> WHERE ap_url LIKE "%.de";
+------------+--------------------------------------+
| ap_kuerzel | ap_url                               |
+------------+--------------------------------------+
| CGN        | http://www.koeln-bonn-airport.de     |
| TXL        | http://www.berlin-airport.de         |
| DUS        | http://www.duesseldorf-international.de |
| FRA        | http://www.frankfurt-airport.de      |
+------------+--------------------------------------+
```

Hier eine Möglichkeit, um den Flughafen Köln/Bonn zu ermitteln, wenn Sie sich nicht mehr sicher sind, ob `"Köln/Bonn"` oder `"Köln-Bonn"` gespeichert wurde:

```
mysql> SELECT ap_name, ap_zusatz FROM rb_airports
    -> WHERE ap_name LIKE "Köln_Bonn";
+-----------+-----------------+
| ap_name   | ap_zusatz       |
+-----------+-----------------+
| Köln/Bonn | Konrad Adenauer |
+-----------+-----------------+
```

Wenn Sie tatsächlich ein Prozentzeichen oder einen Unterstrich als Zeichen benötigen, müssen Sie ihnen einen Backslash (\) voranstellen. Für den Backslash selbst gilt das Gleiche. Das folgende (etwas konstruierte) Beispiel sucht nach Dateinamen von Stadtbildern, die mindestens einen Unterstrich enthalten:

```
SELECT st_name, st_bild_url FROM rb_staedte
WHERE st_bild_url LIKE "%\_%";
```

Die Umkehrung von `LIKE` ist der Operator `NOT LIKE`: Er wählt alle Datensätze aus, auf die der Mustervergleich *nicht* zutrifft. Das folgende Beispiel wählt die Namen aller Städte aus, die kein *e* enthalten:

```
mysql> SELECT st_name FROM rb_staedte
    -> WHERE st_name NOT LIKE "%e%";
+---------------+
| st_name       |
+---------------+
| Dublin        |
| Frankfurt/Main |
| Istanbul      |
```



```
| Köln           |
| Lissabon       |
| London         |
| Lyon           |
| Madrid         |
| Paris          |
| Prag           |
| Rom            |
| Warschau       |
| Zürich         |
+----------------+
```



Die Klausel `LIKE` können Sie übrigens auch bei allen `SHOW ...`-Anweisungen einsetzen, um die Ergebnisse auf das angegebene Muster zu beschränken. Das folgende Beispiel zeigt nur diejenigen Tabellen der Datenbank *reisebuero* an, deren Namen die Zeichenkombination `"flu"` enthalten:

```
mysql> SHOW TABLES FROM reisebuero LIKE "%flu%";
+-----------------------------+
| Tables_in_reisebuero (%flu%) |
+-----------------------------+
| rb_fluege                   |
| rb_flugstrecken             |
+-----------------------------+
```

Reguläre Ausdrücke mit REGEXP

Erheblich mehr Möglichkeiten bietet die Suche nach *regulären Ausdrücken* (engl. *regular expressions*). Das Konzept dieser mächtigen Suchmuster wurde von dem US-Linguisten *Noam Chomsky* entworfen und diente ursprünglich der Beschreibung natürlicher Sprachen. In der Informatik wurden die regulären Ausdrücke vor allem im Unix-Bereich populär; Dienstprogramme wie *sed*, *awk* und *grep* machen ebenso regen Gebrauch von ihnen wie die Programmiersprache *Perl*. PHP ist gleich mit mehreren RegExp-Implementierungen ausgestattet. Die leistungsfähigste (und in diesem Buch ausschließlich behandelte) ist die Perl-kompatible Variante, die mehrere Funktionen mit dem Präfix `preg_` zur Verfügung stellt.

In MySQL werden reguläre Ausdrücke in der Implementierung von *Henry Spencer* verwendet; diese Bibliothek ist POSIX-konform und damit nicht ganz so leistungsfähig wie Perl-kompatible reguläre Ausdrücke. In der Regel ist der Leistungsumfang aber bei Weitem ausreichend für Datenbankoperationen. Tabelle 6-2 zeigt zunächst die wichtigsten RegExp-Konstrukte in MySQL mit Beispielen.

Tabelle 6-2: Von MySQL unterstützte Konstrukte für reguläre Ausdrücke

<table>
<tr><th>Konstrukt</th><th>Bedeutung</th><th>Beispiel</th></tr>
<tr><td>[abc]</td><td>eines der angegebenen Zeichen</td><td>"[HM]aus" findet "Haus" und "Maus", aber nicht "raus"</td></tr>
<tr><td>[^abc]</td><td>beliebiges Zeichen mit Ausnahme der angegebenen</td><td>"[^h]auen" findet "kauen" und "bauen", aber nicht "hauen"</td></tr>
<tr><td>[a-z]</td><td>Zeichenbereich</td><td>"[1-5]000" findet z.B. "1000" oder "4000", aber nicht "6000"</td></tr>
<tr><td>[am-z]</td><td>einzelne Zeichen und Zeichenbereich</td><td>"[14-7]00" findet z.B. "100" oder "500", aber nicht "300" oder "800"</td></tr>
<tr><td>.</td><td>ein beliebiges Zeichen</td><td>".aus" findet "Maus", "Haus" oder "raus", aber nicht "aus"</td></tr>
<tr><td>?</td><td>vorheriger Teilausdruck optional</td><td>"r?aus" findet "raus" oder "aus"</td></tr>
<tr><td>+</td><td>vorheriger Teilausdruck mindestens einmal</td><td>"1+0" findet "10", "110" oder "11111110", aber nicht "0"</td></tr>
<tr><td>*</td><td>vorheriger Teilausdruck beliebig oft</td><td>"10*" findet "1", "10" oder "1000000"</td></tr>
<tr><td>{n}</td><td>vorheriger Teilausdruck genau n-mal</td><td>"[0-9]{5}" findet deutsche Postleitzahlen</td></tr>
<tr><td>{m,n}</td><td>vorheriger Teilausdruck mindestens m-, höchstens n-mal</td><td>"20{2,4}" findet "200", "2000" oder "20000"</td></tr>
<tr><td>(Ausdruck)</td><td>dient der Gruppierung des enthaltenen Teilausdrucks</td><td>"(ha)+" findet z.B. "ha" oder "hahaha"</td></tr>
<tr><td>|</td><td>alternative Gruppen</td><td>"(Köln)|(Bonn)|(Düsseldorf)" findet jede der drei Städte</td></tr>
<tr><td>[:Klasse:]</td><td>findet Zeichen einer bestimmten Gruppe (vollständige Liste in Tabelle 6-3)</td><td>[:alpha:] findet einen beliebigen Buchstaben</td></tr>
<tr><td>^</td><td>Beginn des Strings</td><td>"^er" findet "erleben", aber nicht "aber"</td></tr>
<tr><td>$</td><td>Ende des Strings</td><td>"er$" findet "aber", aber nicht "erleben"</td></tr>
<tr><td>[[:<:]]</td><td>Wortanfang</td><td>"[[:<:]]welt" findet "hallo welt", aber nicht "unterwelt"</td></tr>
<tr><td>[[:>:]]</td><td>Wortende</td><td>"haus[[:>:]]" findet "ins haus gehen", aber nicht "nach hause gehen"</td></tr>
</table>

In Tabelle 6-3 finden Sie sämtliche Zeichenklassen für [:*Klasse*:]-Konstrukte.

Tabelle 6-3: Zeichenklassen zur Verwendung in regulären Ausdrücken

Klasse	Bedeutung
[:alnum:]	alphanumerische Zeichen (Ziffern und Buchstaben)
[:alpha:]	alphabetische Zeichen
[:blank:]	Whitespace (Leerzeichen, Tabulatoren)
[:cntrl:]	Steuerzeichen
[:digit:]	Ziffern

Tabelle 6-3: Zeichenklassen zur Verwendung in regulären Ausdrücken (Fortsetzung)

Klasse	Bedeutung
`[:graph:]`	Grafikzeichen
`[:lower:]`	Kleinbuchstaben
`[:print:]`	Grafik- und Leerzeichen
`[:punct:]`	Interpunktion
`[:space:]`	Leerzeichen, Tabulatoren, Zeilenumbrüche
`[:upper:]`	Großbuchstaben
`[:xdigit:]`	Hexadezimalziffern, also `[0-9A-F]`

Hier einige konkrete Beispiele für die Arbeit mit regulären Ausdrücken:

- `"M[ae][iy]e?r"` passt auf alle Schreibweisen von Namen, die *Meier* ausgesprochen werden: ein *M*, ein *a* oder *e*, ein *i* oder *y*, ein *e* oder auch nicht und zum Schluss ein *r*.
- `"0[1-9][0-9]{1,3}"` (0, genau eine Ziffer zwischen 1 und 9, eine bis drei weitere beliebige Ziffern) findet deutsche Telefonvorwahlen.
- `"[0-9]\-[0-9]{5}\-[0-9]{3}\-[0-9X]"` trifft auf das klassische zehnstellige Format der ISBN (internationale Buchbestellnummern) zu: eine Ziffer, Strich, fünf Ziffern, Strich, drei Ziffern, Strich, Ziffer oder X. Da manche Buchhändler – zum Beispiel Amazon – die Striche weglassen, könnte es nützlich sein, sie optional zu machen: `"[0-9]\-?[0-9]{5}\-?[0-9]{3}\-?[0-9X]"`.

  Die seit 2007 bevorzugt verwendeten 13-stelligen ISBN haben dagegen das Format `"[0-9]{3}\-?[0-9]\-?[0-9]{5}\-?[0-9]{3}\-?[0-9]"`; die Prüfziffer `X` kann bei ihnen nicht vorkommen.



Der Backslash vor jedem der Striche (`\-`) ist wichtig: Zeichen, die in regulären Ausdrücken eine besondere Bedeutung haben, müssen so gekennzeichnet werden, falls Sie das Zeichen als solches meinen. Dazu gehören vor allem folgende Zeichen: `. ? + * [ ] ( ) { } | - \`

- `"^M"` ist die RegExp-Version einer Suche nach allen Feldern, die mit *M* beginnen. Hier als Beispiel die Namen der Städte:

```
mysql> SELECT st_name FROM rb_staedte
    -> WHERE st_name REGEXP "^M";
+---------+
| st_name |
+---------+
| Madrid  |
+---------+
```

- "^[A-E]" wählt alle Felder aus, die mit einem der Buchstaben *A* bis *E* anfangen. Auf die Eigennamen der Städte wirkt sich das wie folgt aus:

```
mysql> SELECT st_eigen FROM rb_staedte
    -> WHERE st_eigen REGEXP "^[A-E]";
+------------+
| st_eigen   |
+------------+
| Amsterdam  |
| Athina     |
| Barcelona  |
| Berlin     |
| Bruxelles  |
| Dublin     |
| Düsseldorf |
| Edinburgh  |
+------------+
```

Beachten Sie, dass reguläre Ausdrücke in MySQL seit Version 3.23.4 nicht mehr zwischen Groß- und Kleinschreibung unterscheiden. Sollte das erforderlich sein, müssen Sie das entsprechende Feld mithilfe der Funktion BINARY() ausdrücklich als binär betrachten. Hier sehen Sie beispielsweise eine Abfrage für die Namen aller Städte, die ein *E* enthalten:

```
mysql> SELECT st_name FROM rb_staedte
    -> WHERE st_name REGEXP "E";
+------------+
| st_name    |
+------------+
| Amsterdam  |
| Athen      |
| Barcelona  |
| Berlin     |
| Brüssel    |
| Düsseldorf |
| Edinburgh  |
| Venedig    |
| Wien       |
+------------+
```

Hier nun die BINARY-Variante, die Sie verwenden müssen, wenn Sie ausdrücklich ein *großes E* meinen:

```
mysql> SELECT st_name FROM rb_staedte
    -> WHERE BINARY(st_name) REGEXP "E";
+-----------+
| st_name   |
+-----------+
| Edinburgh |
+-----------+
```



Ich empfehle Ihnen das hervorragende Buch *Reguläre Ausdrücke* von Jeffrey E. F. Friedl (O'Reilly Verlag), wenn Sie mehr über dieses faszinierende Thema wissen möchten. In diesem Buch wird das Thema nicht nur gründlich und in diversen Implementierungen für verschiedene Programmiersprachen und Umgebungen, sondern auch überaus unterhaltsam präsentiert. Wenn Sie ausreichend Englisch können, hilft auch die eingebaute Hilfe der Programmiersprache Perl weiter. Geben Sie dazu an der Konsole Folgendes ein:

```
$ perldoc perlre
```

Beachten Sie aber, dass zahlreiche fortgeschrittene Features der Perl-RegExp in MySQL nicht zur Verfügung stehen.

Kriterien verknüpfen

Viele Abfragen müssen nicht nur einem, sondern mehreren Auswahlkriterien genügen. Zu diesem Zweck existieren logische Operatoren zur Verknüpfung: Wenn Sie Ausdrücke mithilfe von `AND` oder `&&` verbinden, müssen *beide* wahr sein, damit ein Datensatz ausgewählt wird. `OR` oder `||` wählt dagegen alle Datensätze aus, bei denen *mindestens ein* Kriterium zutrifft. Hier ein erstes `OR`-Beispiel; es wählt alle Städte aus, deren Namen mit *A* oder *K* beginnen:

```
mysql> SELECT st_name, st_eigen FROM rb_staedte
    -> WHERE st_name LIKE "A%" OR st_name LIKE "K%";
+-----------+-----------+
| st_name   | st_eigen  |
+-----------+-----------+
| Amsterdam | Amsterdam |
| Athen     | Athina    |
| Köln      | Köln      |
+-----------+-----------+
```

Es folgt ein `AND`-Beispiel – alle Hotels, in denen ein Doppelzimmer unter 130 € kostet *und* zu deren Zimmern ein komplettes Bad gehört:

```
mysql> SELECT ht_name, ht_dzpreis FROM rb_hotels
    -> WHERE ht_dzpreis < 130 AND ht_bad="Bad";
+-------------------+------------+
| ht_name           | ht_dzpreis |
+-------------------+------------+
| Hotel Apollon     |        120 |
| Hotel Solidarnosc |         70 |
| Finnegan's        |        120 |
+-------------------+------------+
```

Der (vorangestellte) logische Operator `NOT` oder `!` bedeutet, dass das Gegenteil des entsprechenden Ausdrucks gelten soll. Das folgende Beispiel wählt alle Hotels aus, in denen die Verpflegung nicht `"Frühstück"` ist:

```
mysql> SELECT ht_name, ht_mahlzeit FROM rb_hotels
    -> WHERE NOT (ht_mahlzeit LIKE "Fr%");
+-----------------------+-------------+
| ht_name               | ht_mahlzeit |
+-----------------------+-------------+
| Hotel Colosseo        | ohne        |
| Old Palace Hotel      | HP          |
| Hotel Konigin Juliana | HP          |
+-----------------------+-------------+
```

Die Klammern sind hier notwendig, weil `NOT` eine hohe Priorität besitzt und daher meist nur den unmittelbar dahinter stehenden Operanden betrifft – in diesem Fall würde also *ht_mahlzeit* negiert, was keinen Sinn ergibt. Übrigens können Sie überall dort Klammern verwenden, wo die Rangfolge von Operatoren verändert werden muss. Ein kleines Beispiel aus der Arithmetik (Punkt- vor Strichrechnung):

```
mysql> SELECT 3 + 5 * 9, (3 + 5) * 9;
+-----------+-------------+
| 3 + 5 * 9 | (3 + 5) * 9 |
+-----------+-------------+
|        48 |          72 |
+-----------+-------------+
```

Seit MySQL 4.0 steht der besondere Operator `XOR` (*Exklusiv Oder*) zur Verfügung. Der verknüpfte Ausdruck ist hier nur dann wahr, wenn *genau ein* Teilausdruck stimmt. Hier also alle Hotels, die *entweder* nur eine Dusche *oder* nur Frühstück (aber eben nicht beides) anbieten:

```
mysql> SELECT ht_name, ht_bad, ht_mahlzeit FROM rb_hotels
    -> WHERE ht_bad="Dusche" XOR ht_mahlzeit LIKE "Fr%";
+-------------------+--------+-------------+
| ht_name           | ht_bad | ht_mahlzeit |
+-------------------+--------+-------------+
| Hotel Colonia     | Bad    | Frühstück   |
| Hotel au Jardin   | Bad    | Frühstück   |
| Hotel Colosseo    | Dusche | ohne        |
| Queen Victoria    | Bad    | Frühstück   |
| Hotel Cervantes   | Bad    | Frühstück   |
| La Barca          | Bad    | Frühstück   |
| Hotel Apollon     | Bad    | Frühstück   |
| Hotel Royal       | Bad    | Frühstück   |
| Hotel Solidarnosc | Bad    | Frühstück   |
| Haus Alpenhof     | Bad    | Frühstück   |
| Hotel San Marco   | Bad    | Frühstück   |
| Finnegan's        | Bad    | Frühstück   |
+-------------------+--------+-------------+
```

Die Verwendung des gewöhnlichen `OR` bringt dagegen ein anderes Ergebnis, da diese Abfrage auch Datensätze einschließt, in denen beides zutrifft (hier auf die ersten sechs Ergebnisse beschränkt):

```
mysql> SELECT ht_name, ht_bad, ht_mahlzeit FROM rb_hotels
    -> WHERE ht_bad="Dusche" OR ht_mahlzeit LIKE "Fr%"
    -> LIMIT 0,6;
+------------------+--------+-------------+
| ht_name          | ht_bad | ht_mahlzeit |
+------------------+--------+-------------+
| Bergerhof        | Dusche | Frühstück   |
| Hotel Colonia    | Bad    | Frühstück   |
| Hotel de la Gare | Dusche | Frühstück   |
| Hotel au Jardin  | Bad    | Frühstück   |
| Otel Bahar       | Dusche | Frühstück   |
| Hotel Colosseo   | Dusche | ohne        |
+------------------+--------+-------------+
```

Wieder anders verhält sich selbstverständlich die `AND`-Verknüpfung, in der beide Bedingungen zutreffen müssen – im Grunde bildet `OR` die Vereinigungsmenge von `AND` und `XOR`.

Joins

Der wichtigste Grund für die Verwendung relationaler Datenbanken ist die Verknüpfung von Daten mehrerer Tabellen. Praktische Beispiele dafür haben Sie in den vorigen Kapiteln bereits gesehen; hier geht es dagegen um die allgemeine Funktionsweise der Joins.

Zunächst einmal ist es wichtig zu verstehen, dass es in den meisten Fällen keinen Sinn ergibt, Daten aus mehreren Tabellen auszuwählen, ohne anzugeben, wie sie miteinander verknüpft werden sollen. Das ergibt nämlich *jede beliebige Kombination*; der Fachausdruck für eine solche unspezifizierte Verknüpfung lautet *Cross Join* oder *kartesisches Produkt*. Stellen Sie sich dazu vor, Sie hätten drei Tabellen namens *suppe*, *hauptgericht* und *nachtisch* mit je einer gleichnamigen Spalte, die die folgenden Inhalte aufweisen:

```
mysql> SELECT * FROM suppe;
+-------------------+
| suppe             |
+-------------------+
| Tomatensuppe      |
| Spargelcremesuppe |
| Zwiebelsuppe      |
+-------------------+

mysql> SELECT * FROM hauptgericht;
+-----------------------------+
| hauptgericht                |
+-----------------------------+
| Schnitzel                   |
| Bauernomelett               |
| Pizza Spinaci               |
+-----------------------------+
```

```
mysql> SELECT * FROM nachtisch;
+--------------------+
| nachtisch          |
+--------------------+
| Mousse au Chocolat |
| Vanillepudding     |
+--------------------+
```

In einem solchen eher ungewöhnlichen Fall ist der Cross Join sinnvoll, weil sich auf diese Weise jedes mögliche Menü kombinieren lässt:

```
mysql> SELECT * FROM suppe, hauptgericht, nachtisch;
+-------------------+----------------+--------------------+
| suppe             | hauptgericht   | nachtisch          |
+-------------------+----------------+--------------------+
| Tomatensuppe      | Schnitzel      | Mousse au Chocolat |
| Spargelcremesuppe | Schnitzel      | Mousse au Chocolat |
| Zwiebelsuppe      | Schnitzel      | Mousse au Chocolat |
| Tomatensuppe      | Bauernomelett  | Mousse au Chocolat |
| Spargelcremesuppe | Bauernomelett  | Mousse au Chocolat |
| Zwiebelsuppe      | Bauernomelett  | Mousse au Chocolat |
| Tomatensuppe      | Pizza Spinaci  | Mousse au Chocolat |
| Spargelcremesuppe | Pizza Spinaci  | Mousse au Chocolat |
| Zwiebelsuppe      | Pizza Spinaci  | Mousse au Chocolat |
| Tomatensuppe      | Schnitzel      | Vanillepudding     |
| Spargelcremesuppe | Schnitzel      | Vanillepudding     |
| Zwiebelsuppe      | Schnitzel      | Vanillepudding     |
| Tomatensuppe      | Bauernomelette | Vanillepudding     |
| Spargelcremesuppe | Bauernomelette | Vanillepudding     |
| Zwiebelsuppe      | Bauernomelette | Vanillepudding     |
| Tomatensuppe      | Pizza Spinaci  | Vanillepudding     |
| Spargelcremesuppe | Pizza Spinaci  | Vanillepudding     |
| Zwiebelsuppe      | Pizza Spinaci  | Vanillepudding     |
+-------------------+----------------+--------------------+
```

Statt die Tabellen durch Kommata zu trennen, können Sie `[CROSS] JOIN` auch explizit hinschreiben – die beiden folgenden Abfragen sind mit der obigen identisch, so dass das Ergebnis hier nicht noch einmal dargestellt zu werden braucht:

```
SELECT * FROM suppe JOIN hauptgericht JOIN nachtisch;
SELECT * FROM suppe CROSS JOIN hauptgericht CROSS JOIN nachtisch;
```

Normalerweise ist dieses Verhalten aber nicht wünschenswert – beispielsweise möchten Sie sicher nicht jede Stadt mit jedem Flughafen kombinieren, sondern lediglich wissen, in welchen Städten sich die einzelnen Flughäfen befinden. Zu diesem Zweck wird ein sogenannter *Inner Join* der gewünschten Spalten durchgeführt; für das genannte Beispiel wird die Stadtnummer in *rb_airports* mit dem Primärschlüssel von *rb_staedte* verknüpft. Das lässt sich entweder mithilfe einer `WHERE`-Klausel oder durch ein explizites `INNER JOIN` ausdrücken. Hier zunächst die `WHERE`-Schreibweise:

```
mysql> SELECT st_name AS Stadt, ap_name AS Flughafen
    -> FROM rb_staedte, rb_airports
    -> WHERE st_nr=ap_stadt
    -> LIMIT 0,5;
+----------+------------------------+
| Stadt    | Flughafen              |
+----------+------------------------+
| Köln     | Köln/Bonn Airport      |
| Paris    | Paris Charles de Gaulle |
| Istanbul | Istanbul Atatürk       |
| Rom      | Roma Fiumicino         |
| London   | London Heathrow Airport |
+----------+------------------------+
```

Und das sind die nächsten fünf Flughäfen in `INNER JOIN`-Syntax:

```
mysql> SELECT st_name AS Stadt, ap_name AS Flughafen
    -> FROM rb_staedte INNER JOIN rb_airports
    -> ON st_nr=ap_stadt
    -> LIMIT 5,5;
+-----------+---------------------+
| Stadt     | Flughafen           |
+-----------+---------------------+
| Madrid    | Madrid Barajas      |
| Barcelona | Barcelona El Prat   |
| Athen     | Athen International |
| Amsterdam | Amsterdam Schiphol  |
| Brüssel   | Brüssel Zaventem    |
+-----------+---------------------+
```

Wie Sie sehen, wird statt des Kommas `INNER JOIN` zwischen die Tabellennamen in der `FROM`-Klausel gesetzt; das Join-Kriterium `ON` ist mit dem Inhalt einer entsprechenden `WHERE`-Klausel identisch.

Die Besonderheit des Inner Join ist übrigens, dass nur diejenigen Ergebnisse ausgewählt werden, die Daten aus beiden Tabellen enthalten. Bei anderen Join-Typen, die seltener zum Einsatz kommen, ist dies nicht der Fall. Ein *Left Join* wählt auf jeden Fall alle relevanten Datensätze aus derjenigen Tabelle aus, die in der Abfrage links, das heißt vor `LEFT JOIN`, steht – egal ob die rechte Tabelle korrespondierende Daten enthält oder nicht. Vergleichen Sie den folgenden Left Join mit einem Inner Join von *rb_staedte* und *rb_hotels* (nicht für jede Stadt sind Hotels verfügbar):

```
mysql> SELECT st_name, ht_name FROM rb_staedte LEFT JOIN rb_hotels
    -> ON st_nr=ht_stadt ORDER BY st_name LIMIT 0,10;
+----------------+-----------------------+
| st_name        | ht_name               |
+----------------+-----------------------+
| Amsterdam      | Hotel Konigin Beatrix |
| Athen          | Hotel Apollon         |
| Barcelona      | La Barca              |
| Berlin         | Hotel Eichenhof       |
| Brüssel        | Hotel Royal           |
| Dublin         | Finnegan's            |
```

```
| Düsseldorf     | NULL                   |
| Edinburgh      | The New Maitland Hotel |
| Frankfurt/Main | NULL                   |
| Istanbul       | Otel Bahar             |
+----------------+------------------------+

mysql> SELECT st_name, ht_name FROM rb_staedte INNER JOIN rb_hotels
    -> ON st_nr=ht_stadt ORDER BY st_name LIMIT 0,10;
+-----------+------------------------+
| st_name   | ht_name                |
+-----------+------------------------+
| Amsterdam | Hotel Konigin Beatrix  |
| Athen     | Hotel Apollon          |
| Barcelona | La Barca               |
| Berlin    | Hotel Eichenhof        |
| Brüssel   | Hotel Royal            |
| Dublin    | Finnegan's             |
| Edinburgh | The New Maitland Hotel |
| Istanbul  | Otel Bahar             |
| Köln      | Hotel Colonia          |
| Köln      | Bergerhof              |
+-----------+------------------------+
```

Wie Sie sehen, kommen Düsseldorf und Frankfurt am Main in der zweiten Abfrage nicht vor, stattdessen rückt Köln nach.

Ein *Right Join* enthält entsprechend alle Ergebnisse aus der rechten Tabelle, auch wenn diese keine Entsprechung in der linken haben. Wenn Sie es testen möchten, können Sie die obige Abfrage wie folgt umkehren:

```
SELECT st_name, ht_name FROM rb_hotels RIGHT JOIN rb_staedte
ON ht_stadt=st_nr ORDER BY st_name LIMIT 0,10;
```

Interessant ist im Übrigen, dass Sie eine Tabelle auch mit sich selbst verknüpfen können, um innerhalb desselben Ergebnisses Daten aus zwei ihrer Datensätze auszulesen. Dazu müssen Sie der Tabelle allerdings zwei verschiedene Aliasnamen zuweisen und dann stets `Tabellenalias.Feldname` schreiben, um die beiden Verwendungen auseinanderzuhalten.

Hier ein Beispiel, das alle verfügbaren Flugstrecken ab Köln/Bonn (beschränkt auf die ersten zehn), alphabetisch sortiert nach Zielflughäfen, zeigt:

```
mysql> SELECT DISTINCT start.ap_name Ab, ziel.ap_name An
    -> FROM rb_flugstrecken INNER JOIN rb_airports start
    -> INNER JOIN rb_airports ziel
    -> ON fs_start=start.ap_nr AND fs_ziel=ziel.ap_nr
    -> WHERE start.ap_name="Köln/Bonn" ORDER BY ziel.ap_name ASC
    -> LIMIT 0,10;
+-----------+------------------------+
| Ab        | An                     |
+-----------+------------------------+
| Köln/Bonn | Amsterdam Schiphol     |
| Köln/Bonn | Athen International    |
| Köln/Bonn | Berlin Tegel           |
```

```
| Köln/Bonn | Dublin Airport         |
| Köln/Bonn | Edinburgh Airport      |
| Köln/Bonn | Frankfurt Airport      |
| Köln/Bonn | Istanbul Atatürk       |
| Köln/Bonn | Lissabon-Portela       |
| Köln/Bonn | London Heathrow Airport |
| Köln/Bonn | Madrid Barajas         |
+-----------+------------------------+
```

Ergebnisse sortieren – die ORDER BY-Klausel

Mithilfe der bereits erwähnten Klausel `ORDER BY` lassen sich die Ergebnisse von Auswahlabfragen nach einer oder mehreren Spalten sortieren. Die grundsätzliche Syntax dieser Klausel sieht folgendermaßen aus:

```
ORDER BY Feldname [ASC|DESC][, Feldname [ASC|DESC], ...]
```

`ASC` beziehungsweise `DESC` gibt die Sortierrichtung an: `ASC` ist die Abkürzung für *ascending* (aufsteigend); die Richtung ist *1, 2, 3* oder *"a", "b", "c"*. `DESC` (*descending*) gibt dagegen die absteigende Richtung an, also *3, 2, 1* beziehungsweise *"z", "y", "x"*. Die Angabe eines zweiten Felds wird nur dann beachtet, wenn die Spalte mit dem primären Sortierkriterium mehrmals denselben Wert enthält.

Das folgende Beispiel gibt die Vor- und Nachnamen der Kunden aus und sortiert sie aufsteigend nach den Nachnamen:

```
mysql> SELECT kd_vorname, kd_name FROM rb_kunden
    -> ORDER BY kd_name ASC;
+------------+---------+
| kd_vorname | kd_name |
+------------+---------+
| Ludwig     | Becker  |
| Michaela   | Becker  |
| Sven       | Berger  |
| Annette    | Gruber  |
| Michael    | Huber   |
| Siegfried  | Jäger   |
| Klaus      | Klein   |
| Heinrich   | Meier   |
| Klaus      | Schmitz |
| Susanne    | Schmitz |
| Martina    | Vogel   |
+------------+---------+
```

Zufälligerweise sind die Personen mit dem gleichen Nachnamen bereits korrekt nach Vornamen sortiert. Sollte das nicht der Fall sein, könnten Sie es mithilfe der folgenden Abfrage erreichen:

```
SELECT kd_vorname, kd_name FROM rb_kunden
ORDER BY kd_name ASC, kd_vorname ASC;
```

Hier ein weiteres Beispiel – es gibt die fünf teuersten Hotels (Doppelzimmerpreis) mitsamt ihren Städten aus:

```
mysql> SELECT ht_name AS Hotel, st_name AS Stadt, ht_dzpreis AS Preis
    -> FROM rb_hotels INNER JOIN rb_staedte
    -> ON ht_stadt=st_nr
    -> ORDER BY ht_dzpreis DESC
    -> LIMIT 0,5;
+------------------+-----------+-------+
| Hotel            | Stadt     | Preis |
+------------------+-----------+-------+
| Old Palace Hotel | London    |   240 |
| Hotel au Jardin  | Paris     |   200 |
| Hotel Colonia    | Köln      |   195 |
| Queen Victoria   | London    |   190 |
| La Barca         | Barcelona |   185 |
+------------------+-----------+-------+
```

Für Strings gelten auch beim Sortieren die weiter oben beschriebenen Hinweise über Zeichensätze und Kollationen. Falls Sie sich bei einzelnen Sortiervorgängen allein auf den Zeichensatz verlassen möchten, können Sie wiederum die Funktion `BINARY()` verwenden.

SQL-Ausdrücke und -Funktionen

Manchmal ist es erforderlich, aus den Inhalten von Datenbanktabellen andere Werte abzuleiten. Für solche Fälle können `SELECT`-Abfragen nicht nur Felder auslesen, sondern beliebige Ausdrücke enthalten, die berechnet beziehungsweise ausgewertet werden. Beispielsweise könnte sich ein Unternehmen für die Nettopreise der Kölner Hotels (ausgewiesener Endpreis abzüglich Mehrwertsteuer) interessieren. Eine entsprechende Abfrage sähe so aus:

```
mysql> SELECT ht_name AS Hotel, ht_ezpreis / 1.19 AS "EZ Netto",
    -> ht_dzpreis / 1.19 AS "DZ Netto" FROM rb_hotels
    -> WHERE ht_stadt=1;
+---------------+----------+----------+
| Hotel         | EZ Netto | DZ Netto |
+---------------+----------+----------+
| Bergerhof     |  58.8235 | 100.8403 |
| Hotel Colonia |  88.2353 | 163.8655 |
+---------------+----------+----------+
```

Wie Sie sehen, ist die Umbenennung von Ergebnisspalten mittels `AS` hier noch nützlicher als beim Ermitteln normaler Feldinhalte.

Etwas komplexer ist die entsprechende Berechnung für alle deutschen Hotels: In der Tabelle *rb_hotels* ist lediglich ein Verweis auf die jeweilige Stadt in *rb_staedte* gespeichert; diese Tabelle speichert das Land als Relation auf *rb_laender*. Lösen lässt sich das Problem natürlich trotzdem, und zwar mithilfe der folgenden etwas komplexeren Abfrage (wobei in diesem Fall bekannt ist, dass Deutschland durch die Nummer 1 bezeichnet wird):

```
mysql> SELECT ht_name AS Hotel, st_name AS Stadt,
    -> ht_ezpreis / 1.19 AS "EZ Netto",
    -> ht_dzpreis / 1.19 AS "DZ Netto"
    -> FROM rb_hotels INNER JOIN rb_staedte
    -> ON ht_stadt=st_nr AND st_land=1;
+-----------------+--------+----------+----------+
| Hotel           | Stadt  | EZ Netto | DZ Netto |
+-----------------+--------+----------+----------+
| Bergerhof       | Köln   |  58.8235 | 100.8403 |
| Hotel Colonia   | Köln   |  88.2353 | 163.8655 |
| Hotel Eichenhof | Berlin |  67.2269 | 109.2437 |
+-----------------+--------+----------+----------+
```

Neben der hier verwendeten Division stellt MySQL natürlich auch die anderen bekannten arithmetischen Operationen zur Verfügung: Addition (+), Subtraktion (-) und Multiplikation (*). Zusätzlich existiert noch der Operator `DIV` zur ganzzahligen Division. Beachten Sie, dass er nicht nur das Ergebnis, sondern auch die Operanden mathematisch korrekt zu ganzen Zahlen rundet. Daher ist er für die obige Netto-Berechnung nicht geeignet – `120 DIV 1.19` ergibt nicht 103, sondern 120.

Die *Modulo*-Operation (`MOD` oder kurz `%`) liefert den *Rest* einer ganzzahligen Division. In früheren Versionen musste die Funktionsschreibweise `MOD(`*`Argument1`*`, `*`Argument2`*`)` verwendet werden; seit MySQL 4.1 sind die folgenden drei Abfragen äquivalent und ergeben allesamt 4:

```
SELECT MOD(14,5);
SELECT 14 MOD 5;
SELECT 14 % 5;
```

Mathematische Funktionen

MySQL definiert zahlreiche mathematische Funktionen für vielfältige Einsatzzwecke. Tabelle 6-4 zeigt eine Übersicht. Fließkommaergebnisse in den Beispielen wurden hier auf fünf Stellen hinter dem Komma gerundet.

Tabelle 6-4: Die mathematischen Funktionen in MySQL

Funktion	Bedeutung	Beispiel
`ABS(X)`	Absolutwert	`ABS(-2) -> 2`
`ACOS(X)`	Arcus-Cosinus (Cosinus-Umkehrung)[a]	`ACOS(1) -> 0`
`ASIN(X)`	Arcus-Sinus	`ASIN(0) -> 0`
`ATAN(X)`	Arcus-Tangens	`ATAN(3) -> 1.24905`
`CEILING(X) | CEIL(X)`	nächsthöhere Ganzzahl	`CEIL(3.1) -> 4`
`COS(X)`	Cosinus	`COS(0) -> 1`
`COT(X)`	Cotangens	`COT(2) -> -0.45766`

Tabelle 6-4: Die mathematischen Funktionen in MySQL (Fortsetzung)

Funktion	Bedeutung	Beispiel
`CRC32(`*`Str`*`)`	CRC-Prüfsumme (Cyclic Redundancy Check)	`CRC32('Hallo Welt')` `-> 1344803957`
`DEGREES(`*`X`*`)`	Bogenmaß -> Grad	`DEGREES(PI()) -> 180`
`EXP(`*`X`*`)`	e^X	`EXP(1) -> 2.71828`
`FLOOR(`*`X`*`)`	nächstniedrigere Ganzzahl	`FLOOR(4.9) -> 4`
`LN(`*`X`*`)`	natürlicher Logarithmus	`LN(3) -> 1.09861`
`LOG(`*`B`*`,`*`X`*`)`	Logarithmus zur Basis *B* (entspricht `LN()`, wenn *B* weggelassen wird)	`LOG(4,8) -> 1.5`
`LOG2(`*`X`*`)`	Logarithmus zur Basis 2	`LOG2(1024) -> 10`
`LOG10(`*`X`*`)`	Logarithmus zur Basis 10	`LOG10(100000) -> 5`
`PI()`	der Wert von π	`PI() -> 3.14159`
`POW(`*`X`*`,`*`Y`*`) | POWER(`*`X`*`,`*`Y`*`)`	X^Y	`POW(3,4) -> 81`
`RADIANS(`*`X`*`)`	Grad -> Bogenmaß	`RADIANS(180) -> 3.14159`
`RAND(), RAND(`*`N`*`)`	Zufallszahl (mit Random-Seed *N*: stets gleiches Ergebnis)	`RAND() -> 0.23998` `RAND(4) -> 0.15595 (immer)`
`ROUND(`*`X`*`) | ROUND(`*`X`*`,`*`D`*`)`	runden (auf *D* Nachkommastellen)	`ROUND(1.88) -> 2` `ROUND(1.88,1) -> 1.9`
`SIGN(`*`X`*`)`	Vorzeichen (als `-1`, `0` oder `1`)	`SIGN(-7) -> -1` `SIGN(3) -> 1`
`SIN(`*`X`*`)`	Sinus	`SIN(PI()/2) -> 1`
`SQRT(`*`X`*`)`	Quadratwurzel	`SQRT(256) -> 16`
`TAN(`*`X`*`)`	Tangens	`TAN(3) -> -0.14255`
`TRUNCATE(`*`X`*`,`*`D`*`)`	nach *D* Nachkommastellen abschneiden	`TRUNCATE(1.88,1) -> 1.8`

[a] Alle trigonometrischen Funktionen nehmen Winkel im Bogenmaß entgegen; ebenso liefern ihre Umkehrfunktionen Bogenmaß als Ergebnis. Im Bogenmaß gilt: 360° = 2π.

Zwischen dem Funktionsnamen und der öffnenden Klammer darf – anders als in vielen Programmiersprachen – *kein* Leerzeichen stehen; andernfalls erhalten Sie eine Fehlermeldung.

Oft werden mehrere mathematische Funktionen miteinander kombiniert, um sinnvolle Ausdrücke zu bilden. Das folgende Beispiel simuliert mithilfe von `RAND()` und `CEIL()` drei Würfe eines Würfels:

```
mysql> SELECT CEIL(RAND()*6) AS Wurf1,
    -> CEIL(RAND()*6) AS Wurf2,
    -> CEIL(RAND()*6) AS Wurf3;
+-------+-------+-------+
| Wurf1 | Wurf2 | Wurf3 |
+-------+-------+-------+
|     1 |     6 |     5 |
+-------+-------+-------+
```

RAND() bietet einen interessanten zusätzlichen Nutzen: Sie können Tabellen nach Zufall sortieren, indem Sie die Klausel ORDER BY RAND() einsetzen. Hier ein Beispiel, das zweimal fünf zufällige Städte auswählt:

```
mysql> SELECT st_name FROM rb_staedte ORDER BY RAND() LIMIT 0,5;
+-----------+
| st_name   |
+-----------+
| Paris     |
| Dublin    |
| Köln      |
| Venedig   |
| Barcelona |
+-----------+

mysql> SELECT st_name FROM rb_staedte ORDER BY RAND() LIMIT 0,5;
+----------------+
| st_name        |
+----------------+
| Frankfurt/Main |
| Köln           |
| Edinburgh      |
| Warschau       |
| Barcelona      |
+----------------+
```

Einige der mathematischen Funktionen, zum Beispiel die Rundungsfunktionen, werden in den Praxisbeispielen im übernächsten Kapitel zum Einsatz kommen, andere sind dagegen eher exotisch oder haben mit der Reisebüro-Anwendung nichts zu tun.

String-Funktionen

Zur Verarbeitung von Strings gibt es noch mehr Funktionen als für die Mathematik. Tabelle 6-5 zeigt nur die wichtigsten, da einige Funktionen sehr spezielle Aufgaben erfüllen.

Tabelle 6-5: Die wichtigsten String-Funktionen von MySQL

Funktion	Bedeutung	Beispiel
BIN(*N*)	Binärwert der Zahl *N*	BIN(76) -> "1001100"
CO-CAT(*String1*,*String2*, ...)	Verknüpfung	CONCAT("Hallo", " ","Welt") -> "Hallo Welt"
CONV(*N*,*Basis1*,*Basis2*)	konvertiert Zahl *N* von *Basis1* nach *Basis2* (je 2 bis 36)	CONV("AB",16,8) -> 253 CONV("AB",16,10) -> 171
HEX(*N*)	konvertiert *N* in einen hexadezimalen Wert	HEX(154) -> "9A"

Tabelle 6-5: Die wichtigsten String-Funktionen von MySQL (Fortsetzung)

Funktion	Bedeutung	Beispiel
`INSTR(String,Teilstring)`	Position von *Teilstring* in *String* (oder 0)	`INSTR("Hallo Welt","Welt") -> 7`
`LCASE(String) | LOWER(String)`	Umwandlung in Kleinbuchstaben	`LOWER("KLEINE Buchstaben") -> "kleine buchstaben"`
`LENGTH(String)`	Länge des Strings	`LENGTH("Köln") -> 4`
`LTRIM(String)`	führende Leerzeichen entfernen	`LTRIM("      Text") -> "Text"`
`REVERSE(String)`	String umkehren	`REVERSE("lager") -> "regal"`
`RTRIM(String)`	abschließende Leerzeichen entfernen	`RTRIM("Hallo     ") -> "Hallo"`
`SPACE(N)`	*N* Leerzeichen	`SPACE(7) -> "       "`
`SUBSTRING(String, Pos[,N])`	*N* Zeichen langer Teilstring ab *Pos* (ohne *N*: bis Ende)	`SUBSTRING("Kölner",2,2) -> "öl"`
`UCASE(String) | UPPER(String)`	Umwandlung in Großbuchstaben	`UPPER("grosse Buchstaben") -> "GROSSE BUCHSTABEN"`

Selbstverständlich können all diese Funktionen mit Abfrageergebnissen kombiniert werden, was in der Praxis auch meistens der Fall ist. Das folgende Beispiel zeigt den anders lautenden Eigennamen der entsprechenden Städte in Klammern hinter dem deutschen Namen an:

```
mysql> SELECT CONCAT(st_name," (",st_eigen,")") AS Stadt
    -> FROM rb_staedte
    -> WHERE st_name != st_eigen;
+----------------------+
| Stadt                |
+----------------------+
| Rom (Roma)           |
| Athen (Athina)       |
| Brüssel (Bruxelles)  |
| Warschau (Warszawa)  |
| Prag (Praha)         |
| Venedig (Venezia)    |
| Lissabon (Lisboa)    |
+----------------------+
```

Datums- und Uhrzeitfunktionen

Es gibt zwei wichtige Arten von Funktionen zur Datums- und Uhrzeitverarbeitung: Extrahierungsfunktionen, die einzelne Bestandteile von Zeitpunkten ermitteln, und Rechenfunktionen, die zum Berechnen von Zeitintervallen eingesetzt werden. Da MySQL unzählige Funktionen für Datum und Uhrzeit kennt, wird hier pro Kategorie nur je eine besonders nützliche vorgestellt: `DATE_ADD()` und `DATE_FORMAT()`. Darü-

ber hinaus haben Sie bereits die argumentlose Funktion `NOW()` kennengelernt, die die aktuelle Systemzeit im Format `"JJJJ-MM-TT hh:mm:ss"` (zum Beispiel `"2007-04-02 17:32:51"`) ausgibt.

`DATE_ADD(`*`Datum`*`,INTERVAL `*`Anzahl Typ`*`)` addiert eine Zeitspanne zum angegebenen Datum. Das folgende Beispiel rechnet vom 02.04.2007 an drei Tage weiter:

```
mysql> SELECT DATE_ADD('2007-04-02',INTERVAL 3 DAY);
+--------------------------------------+
| DATE_ADD('2007-04-02',INTERVAL 3 DAY) |
+--------------------------------------+
| 2007-04-05                            |
+--------------------------------------+
```

Um eine Zeitspanne abzuziehen, können Sie der Anzahl ein Minuszeichen voranstellen oder die Variante `DATE_SUB()` verwenden. Die beiden folgenden Beispiele sind äquivalent; sie ziehen vom 02.04. zwei Monate ab:

```
mysql> SELECT DATE_ADD('2007-04-02',INTERVAL -2 MONTH);
+-----------------------------------------+
| DATE_ADD('2007-04-02',INTERVAL -2 MONTH) |
+-----------------------------------------+
| 2007-02-02                               |
+-----------------------------------------+

mysql> SELECT DATE_SUB('2007-04-02',INTERVAL 2 MONTH);
+----------------------------------------+
| DATE_SUB('2007-04-02',INTERVAL 2 MONTH) |
+----------------------------------------+
| 2007-02-02                              |
+----------------------------------------+
```

Die wichtigsten Arten von Zeitintervallen sind: `SECOND` (Sekunden), `MINUTE` (Minuten), `HOUR` (Stunden), `DAY` (Tage), `MONTH` (Monate) und `YEAR` (Jahre). Erst seit MySQL 5.0 stehen `WEEK` (Wochen) und `QUARTER` (Quartale) zur Verfügung.

`DATE_FORMAT(`*`Datum`*`,`*`Formatstring`*`)` ermöglicht die beliebige Formatierung einer Zeitangabe oder ihrer Bestandteile. Diese SQL-Funktion ähnelt der PHP-Funktion `date()` oder dem gleichnamigen Unix-Systembefehl; ihre Kenntnis ersetzt im Grunde alle einzelnen Extrahierungsfunktionen wie `DAY(Datum)` zur Ermittlung des Tages im Monat. Tabelle 6-6 führt alle wichtigen Platzhalter für Formatstrings auf.

Tabelle 6-6: Platzhalter für Datumsformate in MySQL

Platzhalter	Bedeutung
%a	Wochentag, Abkürzung (*Sun* bis *Sat*)
%b	Monat, Abkürzung (*Jan* bis *Dec*)
%c	Monat, numerisch (0 bis 12)
%D	Tag mit englischem Suffix (*1st*, *2nd*, *3rd* usw.)
%d	Tag des Monats, numerisch, zweistellig (00 bis 31)

Tabelle 6-6: Platzhalter für Datumsformate in MySQL (Fortsetzung)

Platzhalter	Bedeutung
`%e`	Tag des Monats, numerisch (0 bis 31)
`%f`	Mikrosekunden (000000 bis 999999)
`%H`	Stunde, zweistellig, 24h (00 bis 23)
`%h` oder `%I`	Stunde, zweistellig, 12h (01 bis 12)
`%i`	Minuten, zweistellig (00 bis 59)
`%j`	Tag im Jahr (001 bis 366)
`%k`	Stunde, 24h (0 bis 23)
`%l`	Stunde, 12h (1 bis 12)
`%M`	Monatsname (*January* bis *December*)
`%m`	Monat, numerisch, zweistellig (00 bis 12)
`%p`	*AM* oder *PM*
`%r`	Zeit, 12h mit AM/PM (z.B. *03:59:42 PM*)
`%S` oder `%s`	Sekunden (00 bis 59)
`%T`	Zeit, 24h (z.B. *15:59:42*)
`%U`	Woche ab Sonntag (00 bis 53)
`%u`	Woche ab Montag (00 bis 53)
`%W`	Wochentag (*Sunday* bis *Saturday*)
`%w`	Wochentag, numerisch (0 = Sonntag bis 6 = Samstag)
`%Y`	Jahr, vierstellig (z.B. 1998, 2007)
`%y`	Jahr, zweistellig (z.B. 98, 07)
`%%`	Prozentzeichen

Hier zwei Beispiele für die Formatierung des aktuellen Datums:

```
SELECT DATE_FORMAT(NOW(),"%d.%m.%Y, %H:%i Uhr");
```

Ergebnis: *02.04.2007, 17:05 Uhr*

```
SELECT DATE_FORMAT(NOW(),"%W, %M %D, %Y");
```

Ergebnis: *Monday, April 2nd, 2007*

Wochentag- und Monatsnamen stehen leider nur in englischer Sprache zur Verfügung. Wenn Sie deutsche Namen verwenden möchten, können Sie sich Datenbanktabellen anlegen, die die Nummern den entsprechenden Namen zuordnen. Hier ein Beispiel für die Wochentage:

1. Erstellen Sie in der gewünschten Datenbank eine neue Tabelle namens *wtage*:

   ```
   CREATE TABLE wtage (tag_nr INT PRIMARY KEY, tag_name VARCHAR(12));
   ```

2. Füllen Sie *wtage* mit den Zuordnungen – das Format `%w` verlangt, dass die Werte 0 für Sonntag bis 6 für Samstag verwendet werden müssen:

```
INSERT INTO wtage VALUES
(0, 'Sonntag'),
(1, 'Montag'),
(2, 'Dienstag'),
(3, 'Mittwoch'),
(4, 'Donnerstag'),
(5, 'Freitag'),
(6, 'Samstag');
```

3. Nun können Sie die Wochentage einsetzen – beispielsweise in einer komplexen Abfrage wie dieser:

```
mysql> SELECT CONCAT(tag_name,", ",DATE_FORMAT(NOW(),
    -> "%d.%m.%Y, %H:%i")) AS Heute FROM wtage
    -> WHERE tag_nr=(DATE_FORMAT(NOW(),"%w"));
+--------------------------+
| Heute                    |
+--------------------------+
| Montag, 02.04.2007, 17:21 |
+--------------------------+
```

Interessant sind schließlich noch die sogenannten Unix-Timestamps, weil PHP und andere Programmiersprachen Zeitpunkte in diesem Format speichern. Es handelt sich um die Anzahl der Sekunden seit EPOCH, dem »geglätteten« Unix-Erfindungsdatum: 01.01.1970, 00:00 Uhr GMT. MySQL bietet zwei Funktionen zur Umrechnung in dieses Format beziehungsweise wieder zurück.

`UNIX_TIMESTAMP([`*`Datum`*`])` wandelt den angegebenen Zeitpunkt in einen Unix-Timestamp um. Wenn Sie kein Datum angeben, wird die aktuelle Systemzeit verwendet.

```
mysql> SELECT UNIX_TIMESTAMP("2007-04-07 12:31:17");
+-------------------------------------+
| UNIX_TIMESTAMP("2007-04-07 12:31:17") |
+-------------------------------------+
|                          1112869877 |
+-------------------------------------+
```

`FROM_UNIXTIME(UNIX-Timestamp[,Formatstring])` erledigt die umgekehrte Aufgabe: Die angegebene Sekundenzahl seit EPOCH wird in eine Datums- und Uhrzeitangabe umgewandelt. Optional können Sie ein `DATE_FORMAT()`-kompatibles Format angeben. Beispiele:

```
mysql> SELECT FROM_UNIXTIME(1000000000);
+--------------------------+
| FROM_UNIXTIME(1000000000) |
+--------------------------+
| 2001-09-09 03:46:40      |
+--------------------------+

mysql> SELECT FROM_UNIXTIME(1234567890,"%d.%m.%Y, %H:%i");
+------------------------------------------+
| FROM_UNIXTIME(1234567890,"%d.%m.%Y, %H:%i") |
+------------------------------------------+
| 14.02.2009, 00:31                        |
+------------------------------------------+
```

Aggregatfunktionen

In einigen Fällen sollen keine einzelnen Datensätze ermittelt werden, sondern Informationen über eine Gruppe von Datensätzen. Zu diesem Zweck stellt SQL eine Reihe sogenannter *Aggregatfunktionen* zur Verfügung; der Name weist darauf hin, dass sie mehrere Zeilen zusammenfassen.

Hier die wichtigsten im Überblick:

- `SUM(`*`Spaltenname`*`)` – Summe der Spalte
- `MIN(`*`Spaltenname`*`)` – niedrigster Wert der Spalte
- `MAX(`*`Spaltenname`*`)` – höchster Wert der Spalte
- `AVG(`*`Spaltenname`*`)` – Mittelwert der Spalte
- `COUNT(`*`Spaltenname`*`)` – Anzahl der Datensätze

Hier werden beispielsweise der höchste, der niedrigste und der durchschnittliche Preis für ein Einzelzimmer ermittelt:

```
mysql> SELECT MAX(ht_ezpreis), MIN(ht_ezpreis), AVG(ht_ezpreis)
    -> FROM rb_hotels;
+-----------------+-----------------+-----------------+
| MAX(ht_ezpreis) | MIN(ht_ezpreis) | AVG(ht_ezpreis) |
+-----------------+-----------------+-----------------+
|             130 |              40 |         80.8696 |
+-----------------+-----------------+-----------------+
```

Natürlich nützen Höchst- und Tiefstpreise nicht viel, wenn Sie nicht wissen, um welches Hotel es sich jeweils handelt. Allerdings führt der folgende Ansatz zu einer Fehlermeldung:

```
mysql> SELECT ht_name, MIN(ht_ezpreis) FROM rb_hotels;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...
) with no GROUP columns is illegal if there is no GROUP BY clause
```

Da Aggregatfunktionen die Aufgabe haben, Daten zu gruppieren, können sie nicht gemeinsam mit einem einzelnen Wert ausgewählt werden. Die Lösung bringt erst eine Unterabfrage:

```
mysql> SELECT ht_name, ht_ezpreis FROM rb_hotels
    -> WHERE ht_ezpreis=(SELECT MIN(ht_ezpreis) FROM rb_hotels);
+-------------------+------------+
| ht_name           | ht_ezpreis |
+-------------------+------------+
| Hotel Solidarnosc |         40 |
+-------------------+------------+
```

Üblicher – und einfacher – ist es dagegen, die Aggregatfunktionen tatsächlich zur Gruppierung von Informationen einzusetzen. Dazu wird die gewünschte Spalte mithilfe einer `GROUP BY`-Klausel zusammengefasst. Das folgende Beispiel zeigt, wie viele Hotels jeweils über eine bestimmte Badausstattung verfügen:

```
mysql> SELECT ht_bad AS Badausstattung, COUNT(*) AS Anzahl
    -> FROM rb_hotels GROUP BY ht_bad;
+----------------+--------+
| Badausstattung | Anzahl |
+----------------+--------+
| Dusche         |     10 |
| Bad            |     13 |
+----------------+--------+
```

Da COUNT() Datensätze beziehungsweise Ergebniszeilen zählt, spielt es in der Regel keine Rolle, welchen Spaltennamen Sie als Argument angeben; das hier eingesetzte * (alle Spalten) ist meist die einfachste Lösung.

Natürlich lassen sich Aggregatfunktionen auch mit Joins kombinieren. Beispielsweise könnte es interessant sein zu erfahren, wie viele Hotels in den einzelnen Städten gebucht werden können. Dazu muss der zu *ht_stadt* passende Städtename jeweils aus der Tabelle *rb_staedte* ausgelesen werden. Die fertige Abfrage sieht so aus:

```
mysql> SELECT st_name AS Stadt, COUNT(*) AS "Anzahl Hotels"
    -> FROM rb_staedte INNER JOIN rb_hotels ON st_nr=ht_stadt
    -> GROUP BY st_nr
    -> ORDER BY st_name ASC;
+-----------+---------------+
| Stadt     | Anzahl Hotels |
+-----------+---------------+
| Amsterdam |             1 |
| Athen     |             1 |
| Barcelona |             1 |
| Berlin    |             1 |
| Brüssel   |             1 |
| Dublin    |             1 |
| Edinburgh |             1 |
| Istanbul  |             1 |
| Köln      |             2 |
| Lissabon  |             1 |
| London    |             2 |
| Lyon      |             1 |
| Madrid    |             1 |
| Paris     |             2 |
| Prag      |             1 |
| Rom       |             1 |
| Venedig   |             1 |
| Warschau  |             1 |
| Wien      |             1 |
| Zürich    |             1 |
+-----------+---------------+
```

Weitere Abfragetypen

Neben den bereits besprochenen Erstellungs-, Einfüge- und Auswahlabfragen kennt SQL noch einige andere Abfragetypen. Im Wesentlichen handelt es sich um Änderungsabfragen für Daten und Tabellenstrukturen sowie um Löschabfragen.

Datenänderungsabfragen

Um den Inhalt von Datensätzen zu ändern, wird die SQL-Anweisung `UPDATE` verwendet. Im Wesentlichen hat sie die folgende Syntax:

```
UPDATE Tabelle SET Spalte1=Wert1[, Spalte2=Wert2 ...]
[WHERE Kriterium];
```

Sollen alle Datensätze einer Tabelle auf dieselbe Weise geändert werden, kann die Angabe eines Kriteriums mittels `WHERE` entfallen. Solche Fälle sind eher selten. Im Übrigen ist ein `UPDATE` ohne `WHERE` sehr gefährlich, da Sie durch das Überschreiben sämtlicher Datensätze mit einem identischen Spaltenwert wichtige Daten verlieren könnten. Genau aus diesem Grund soll eine Kopie der Tabelle *rb_hotels* angefertigt werden, damit Sie in den restlichen Abschnitten dieses Kapitels nach Belieben damit spielen können. Die folgenden beiden Abfragen erstellen die Kopie und machen aus *ht_nr* wieder einen Primärschlüssel (wenn auch ohne `AUTO_INCREMENT`, das für diesen Zweck nicht nötig ist):

```
mysql> CREATE TABLE rb_hotels2 SELECT * FROM rb_hotels;
Query OK, 23 rows affected (0.18 sec)
Records: 23  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE rb_hotels2 ADD PRIMARY KEY(ht_nr);
Query OK, 23 rows affected (0.30 sec)
Records: 23  Duplicates: 0  Warnings: 0
```

Die Syntax von `ALTER TABLE` zur Änderung der Tabellenstruktur wird im nächsten Abschnitt genauer beleuchtet.

In der neuen Tabelle soll nun als Erstes eingetragen werden, dass *alle* Hotels ab sofort über Zimmer mit richtigem Bad verfügen:

```
mysql> UPDATE rb_hotels2 SET ht_bad="Bad";
Query OK, 10 rows affected (0.04 sec)
Rows matched: 23  Changed: 10  Warnings: 0
```

Die Meldung `Changed: 10` macht deutlich, dass MySQL nur diejenigen Datensätze ändert, in denen der gewünschte Wert nicht bereits besteht. Das hört sich spitzfindig an, sorgt aber in sehr umfangreichen Datenbanken für Geschwindigkeitsvorteile.

In `UPDATE`-Abfragen können Sie auf die bisherigen Werte der Felder zurückgreifen und diese ändern. Das nächste Beispiel reduziert alle Doppelzimmerpreise (in der kopierten Tabelle!) auf 90% ihres bisherigen Werts. Hier zum Vergleich zunächst die alten Werte der ersten fünf Hotels:

```
mysql> SELECT ht_name, ht_dzpreis FROM rb_hotels2 LIMIT 0,5;
+------------------+------------+
| ht_name          | ht_dzpreis |
+------------------+------------+
| Bergerhof        |        120 |
| Hotel Colonia    |        195 |
| Hotel de la Gare |        150 |
| Hotel au Jardin  |        200 |
| Otel Bahar       |         80 |
+------------------+------------+
```

Die Abfrage zur Preisreduktion sieht wie folgt aus:

```
mysql> UPDATE rb_hotels2 SET ht_dzpreis=ht_dzpreis * 0.9;
Query OK, 23 rows affected (0.05 sec)
Rows matched: 23  Changed: 23  Warnings: 0
```

Nach der Änderung lauten die ersten fünf (nach wie vor ganzzahligen) Preise so:

```
mysql> SELECT ht_name, ht_dzpreis FROM rb_hotels2 LIMIT 0,5;
+------------------+------------+
| ht_name          | ht_dzpreis |
+------------------+------------+
| Bergerhof        |        108 |
| Hotel Colonia    |        175 |
| Hotel de la Gare |        135 |
| Hotel au Jardin  |        180 |
| Otel Bahar       |         72 |
+------------------+------------+
```

Viel häufiger kommt es natürlich vor, dass nur bestimmte Datensätze geändert werden sollen. Ihre Auswahl erfolgt über ein WHERE-Kriterium nach demselben Schema wie bei SELECT-Abfragen. Hier ein Beispiel, das die Preise aller Hotels in Köln (Stadt-Nummer 1) halbiert:

```
mysql> UPDATE rb_hotels2 SET ht_ezpreis = ht_ezpreis * 0.5,
    -> ht_dzpreis = ht_dzpreis * 0.5
    -> WHERE ht_stadt=1;
Query OK, 2 rows affected (0.08 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

Ein SELECT mit derselben Einschränkung liefert – zusammen mit der vorherigen Minderung aller Doppelzimmerpreise – folgendes Ergebnis:

```
mysql> SELECT ht_name, ht_ezpreis, ht_dzpreis FROM rb_hotels2
    -> WHERE ht_stadt=1;
+---------------+------------+------------+
| ht_name       | ht_ezpreis | ht_dzpreis |
+---------------+------------+------------+
| Bergerhof     |         35 |         54 |
| Hotel Colonia |         52 |         87 |
+---------------+------------+------------+
```

Hier ein letztes Beispiel – es ändert explizit den Namen eines (kopierten) Hotels:

```
UPDATE rb_hotels2 SET ht_name="Hotel Konigin Beatrix"
WHERE ht_name="Hotel Konigin Juliana";
```

Strukturänderungsabfragen

Komplexer als Abfragen zur Änderung von Tabelleninhalten sind diejenigen, die den Aufbau von Tabellen manipulieren. Die zuständige SQL-Anweisung lautet ALTER TABLE; die wichtigsten Aspekte ihrer Syntax lauten folgendermaßen:

```
ALTER TABLE Tabellenname
  ADD [COLUMN] Spaltenname Typ [Optionen] [FIRST|AFTER Spaltenname]
| DROP [COLUMN] Spaltenname
| CHANGE [COLUMN] Spaltenname NeuerSpaltenname Typ [Optionen]
                                   [FIRST|AFTER Spaltenname]
| ADD INDEX [Indexname] [Indextyp] (Spaltenname, ...)
| DROP INDEX Indexname
| ADD PRIMARY KEY (Spaltenname, ...)
| DROP PRIMARY KEY
| RENAME [TO] NeuerTabellenname
```

Zum Ausprobieren dieser Optionen sollten Sie wieder die im vorigen Abschnitt erstellte Tabellenkopie *rb_hotels2* benutzen.

ADD COLUMN fügt eine zusätzliche Spalte hinzu; das Schlüsselwort COLUMN selbst ist übrigens stets optional. FIRST bedeutet, dass die neue Spalte ganz links eingefügt wird; mit AFTER *Spaltenname* können Sie dagegen eine bestimmte Spalte angeben, hinter der sie erscheinen soll. Wenn Sie keine Angabe machen, wird die neue Spalte ans Ende gesetzt. Das folgende Beispiel fügt hinter *ht_mahlzeit* eine neue Spalte namens *ht_lang* hinzu, ein SET mit den garantiert im jeweiligen Hotel gesprochenen Fremdsprachen:

```
ALTER TABLE rb_hotels2
ADD COLUMN ht_lang SET("de", "en", "fr", "es", "it")
AFTER ht_mahlzeit;
```

Eine DESC[RIBE]-Anweisung (hier manuell gekürzt, damit sie in den Satzspiegel passt) zeigt, dass es funktioniert hat:

```
mysql> DESC rb_hotels2;
+--------------+-------------------+------+-----+---------+-------+
| Field        | Type              | Null | Key | Default | Extra |
+--------------+-------------------+------+-----+---------+-------+
| ht_nr        | int(11)           |      | PRI | 0       |       |
| ht_name      | varchar(70)       | YES  |     | NULL    |       |
| ht_stadt     | int(11)           |      |     | 0       |       |
| ht_ezpreis   | int(11)           |      |     | 0       |       |
| ht_dzpreis   | int(11)           |      |     | 0       |       |
| ht_bad       | enum('ohne',...)  |      |     | ohne    |       |
| ht_mahlzeit  | enum('ohne',...)  |      |     | ohne    |       |
| ht_lang      | set('de','en',...) | YES  |     | NULL    |       |
| ht_anschrift | varchar(100)      |      |     |         |       |
| ht_url       | varchar(50)       |      |     |         |       |
+--------------+-------------------+------+-----+---------+-------+
```

Zum Löschen von Spalten wird `DROP [COLUMN]` eingesetzt. Sie brauchen lediglich den Spaltennamen anzugeben. Das folgende Beispiel löscht die soeben erstellte Spalte *ht_lang* wieder:

```
ALTER TABLE rb_hotels2
DROP COLUMN ht_lang;
```



`DROP COLUMN` löscht eine Spalte auch dann gnadenlos, wenn sie Daten enthält!

Um die diversen Eigenschaften einer Spalte zu modifizieren, wird `CHANGE COLUMN` verwendet. Sie können den Namen, den Datentyp und die Optionen sowie die Position in der Tabelle ändern. Dabei wird stets die gesamte Definition samt neuem Namen angegeben; optional ist nur die Positionsangabe `FIRST` beziehungsweise `AFTER` *`Spaltenname`*. Das folgende Beispiel benennt *ht_anschrift* in *ht_adresse* um:

```
ALTER TABLE rb_hotels2
CHANGE COLUMN ht_anschrift ht_adresse VARCHAR(100);
```

Hier ein Beispiel für eine Typänderung – *ht_name* soll von 60 auf 70 zulässige Zeichen verlängert werden:

```
ALTER TABLE rb_hotels2
CHANGE COLUMN ht_name ht_name VARCHAR(70);
```

Passen Sie bei Typänderungen auf: Wenn der neue Typ kleiner oder weniger präzise ist als der alte, können Daten verloren gehen. Änderungen von Fließkomma- in Ganzzahltypen oder Verkürzungen von Zeichenketten sind daher relativ gefährlich. Das folgende Beispiel ändert den Datentyp von *ht_ezpreis* von `INT` in `TINYINT` (8-Bit-Integer; Wertebereich –128 bis +127):

```
mysql> ALTER TABLE rb_hotels2
    -> CHANGE COLUMN ht_ezpreis ht_ezpreis TINYINT;
Query OK, 23 rows affected (0.22 sec)
Records: 23  Duplicates: 0  Warnings: 0
```

Falls Sie die Preisminderung um 10% aus dem vorigen Abschnitt in Ihrer Version von *rb_hotels2* nicht durchgeführt haben, erhalten Sie eine Warnung, da der teuerste Einzelzimmerpreis (130 €) automatisch auf 127 gesenkt wird.

Schließlich können Sie `CHANGE COLUMN` noch verwenden, um die Position von Spalten in einer Tabelle zu modifizieren. Das folgende Beispiel verschiebt *ht_stadt* hinter *ht_dzpreis*:

```
ALTER TABLE rb_hotels2
CHANGE COLUMN ht_stadt ht_stadt INT AFTER ht_dzpreis;
```

`ALTER TABLE` kann nicht nur Spalten, sondern auch Indizes ändern. `ADD INDEX` fügt einen Index hinzu, `DROP INDEX` löscht ihn wieder; die entsprechende Syntax wurde bereits im Abschnitt »Schlüssel und Indizes« im vorigen Kapitel erläutert. Dasselbe

gilt für Primärschlüssel, die Sie mithilfe von `ADD PRIMARY KEY` *(Spaltenname)* hinzufügen und mittels `DROP PRIMARY KEY` – da es nur einen gibt, ohne Namensangabe – entfernen können.

Sie können auch die Tabelle selbst umbenennen. Das folgende Beispiel gibt *rb_hotels2* den neuen Namen *rb_hotels_neu*:

```
ALTER TABLE rb_hotels2 RENAME rb_hotels_neu;
```

Löschabfragen

Oft müssen Datensätze aus Tabellen entfernt werden. Dazu werden `DELETE`-Abfragen mit folgender Grundsyntax verwendet:

```
DELETE FROM Tabelle
[WHERE Kriterium]
```

Auch hier ist die Angabe eines Kriteriums mithilfe der üblichen `WHERE`-Syntax wichtig, da sonst alle Datensätze aus der Tabelle gelöscht werden. Das folgende Beispiel löscht alle Hotels aus *rb_hotels_neu*, in denen ein Einzelzimmer über 100 € kostet:

```
mysql> DELETE FROM rb_hotels_neu
    -> WHERE ht_ezpreis > 100;
Query OK, 1 row affected (0.10 sec)
```

Das Ergebnis zeigt, dass nur eine Zeile gelöscht wurde.

Wenn Sie explizit alle Datensätze löschen möchten, können Sie entweder

```
DELETE FROM Tabelle;
```

schreiben, oder Sie verwenden die schnellere Variante

```
TRUNCATE Tabelle;
```

Sie können übrigens auch die kopierte Tabelle selbst löschen; nicht mehr benötigte Tabellen sollten stets entfernt werden, da sie Speicher vergeuden und Datenbanken unübersichtlich machen. Führen Sie dazu folgende Abfrage durch:

```
mysql> DROP TABLE rb_hotels_neu;
Query OK, 0 rows affected (0.04 sec)
```

Zu guter Letzt kann sogar eine ganze Datenbank gelöscht werden. Dazu dient eine `DROP DATABASE`-Abfrage. Beispiel:

```
mysql> DROP DATABASE alte_datenbank;
```

In diesem Kapitel:

- Transaktionen
- Views
- Prepared Statements
- Stored Procedures
- Trigger

KAPITEL 7

Fortgeschrittene Datenbankfunktionen

Den Fortschritt verdanken wir Menschen, die Dinge versucht haben, von denen sie gelernt haben, dass sie nicht gehen.

Robert Lembke

Für das Schreiben MySQL-basierter PHP-Webanwendungen genügen normalerweise die im vorigen Kapitel besprochenen SQL-Aspekte. In diesem Kapitel werden einige Features vorgestellt, die in anderen Datenbanksystemen schon länger existieren, in MySQL aber erst in neueren Versionen eingeführt wurden. Da sie für durchschnittliche Webanwendungen keine große Rolle spielen, werden sie nur relativ kurz besprochen.

Transaktionen

In größeren verteilten Anwendungen müssen oft sehr komplexe Vorgänge in Datenbanken abgebildet werden. Für jeden solchen Vorgang wird in der Regel eine längere Abfolge von SQL-Abfragen eingesetzt. Ein Problem ergibt sich, wenn der Vorgang später im Ganzen rückgängig gemacht werden muss, denn dann gilt es, zahlreiche voneinander abhängige Abfragen zu entwirren.

Aus diesem Grund wurde in SQL das Konzept der *Transaktion* eingeführt: Beliebig viele Abfragen lassen sich zu einer Einheit bündeln. Diese kann zum Schluss insgesamt durch ein *Commit* bestätigt oder durch ein *Rollback* zurückgesetzt werden. Ereignisse wie der Verlust einer Datenbankverbindung oder ein Absturz des Servers führen außerdem zu einem automatischen Rollback, so dass die Daten nicht durch unvollständige Operationen inkonsistent werden.

Die Ziele der Transaktionsverarbeitung werden üblicherweise mit dem Akronym *ACID* bezeichnet. Die vier Buchstaben stehen für folgende Aspekte:

Atomicity (Atomarität)

Jede Transaktion ist *atomar* – alle beteiligten SQL-Anweisungen werden zu einer einzigen zusammengefasst, die ausgeführt wird oder auch nicht.

Consistency (Konsistenz)

Nach dem Abschluss einer Transaktion – sei es durch Commit oder Rollback – muss die Datenbank in einem konsistenten Zustand verbleiben; Constraints dürfen nicht verletzt werden.

Isolation

Wenn gleichzeitig mehrere Transaktionen auf derselben Tabelle durchgeführt werden, laufen sie vollkommen geschützt voreinander sowie vor nicht transaktionsorientierten Abfragen ab.

Durability (Dauerhaftigkeit)

Das Ergebnis einer abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank gespeichert.

Einführung

In MySQL werden Transaktionen nur in InnoDB-Tabellen unterstützt. Hinter den Kulissen werden Abfragen bei diesem Tabellentyp sogar stets als Transaktionen ausgeführt. Allerdings befinden sie sich standardmäßig im sogenannten *Autocommit*-Modus, in dem jede einzelne Abfrage eine eigene Transaktion ist, die durch automatisches Commit abgeschlossen wird.

Die SQL-Anweisungen zur gesteuerten Durchführung von Transaktionen sind sehr einfach. Vor der ersten Abfrage, die zu einer Transaktion gehören soll, wird folgende Anweisung benötigt:

```
START TRANSACTION;
```

Anschließend können Sie in der aktuellen Datenbank beliebig viele verschiedene Abfragen auf InnoDB-Tabellen ausführen. Zum Schluss müssen Sie sich entscheiden, ob die Transaktion bestätigt oder zurückgesetzt werden soll. Zum Bestätigen wird die folgende Anweisung verwendet:

```
COMMIT;
```

Soll die Transaktion dagegen rückgängig gemacht werden, lautet die Anweisung wie folgt:

```
ROLLBACK;
```

Eine Alternative zu einem expliziten `START TRANSACTION` besteht darin, Autocommit auszuschalten. Dies geschieht mithilfe der folgenden Anweisung:

```
SET autocommit=false;
```

Danach sind SQL-Anweisungsfolgen auf InnoDB-Tabellen grundsätzlich Transaktionen; jedes `COMMIT` oder `ROLLBACK` startet implizit eine neue Transaktion. Den aktuellen Zustand von `autocommit` können Sie wie folgt abfragen (wobei 1 für `true` und 0 für `false` ausgegeben wird):

```
mysql> SELECT @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
```

Ein weiteres interessantes Konzept innerhalb von Transaktionen sind *Savepoints*. Wenn Sie einen Savepoint setzen, können Sie statt eines vollständigen Rollbacks auch ein Teil-Rollback zu einem bestimmten Savepoint durchführen. Um einen Savepoint zu erstellen, schreiben Sie einfach:

```
SAVEPOINT Name;
```

Wenn Sie später zu diesem Savepoint zurückkehren möchten, funktioniert dies wie folgt:

```
ROLLBACK TO [SAVEPOINT] Name;
```

Praktischer Test

Da in der *reisebuero*-Datenbank keine Transaktionen vorgesehen sind, sollten Sie zuerst eine zusätzliche Testdatenbank erstellen und als Standard auswählen:

```
CREATE DATABASE trans_test;
USE trans_test
```

Nun wird in der neuen Datenbank eine einzelne Tabelle erstellt:

```
CREATE TABLE stationen (
  s_id INT AUTO_INCREMENT PRIMARY KEY,
  s_name VARCHAR (40),
  s_dauer INT
) ENGINE=InnoDB;
```

Vor dem Einfügen des ersten Datensatzes wird eine Transaktion gestartet:

```
START TRANSACTION;
```

Nun werden einige Datensätze hinzugefügt:

```
INSERT INTO stationen (s_name, s_dauer) VALUES
("Köln", 0);
INSERT INTO stationen (s_name, s_dauer) VALUES
("Bonn", 30);
INSERT INTO stationen (s_name, s_dauer) VALUES
("Frankfurt", 80);
```

Eine einfache Auswahlabfrage zeigt, dass alle drei Datensätze tatsächlich vorhanden sind:

```
mysql> SELECT * FROM stationen;
+------+-----------+---------+
| s_id | s_name    | s_dauer |
+------+-----------+---------+
|    1 | Köln      |       0 |
|    2 | Bonn      |      30 |
|    3 | Frankfurt |      80 |
+------+-----------+---------+
```

Öffnen Sie nun ein zweites Terminal- oder Eingabeaufforderungsfenster und starten auch darin den *mysql*-Client als *root* oder als ein anderer User, der vollen Zugriff auf die Datenbank *trans_test* besitzt. Wenn Sie in diesem Fenster alle Datensätze aus der Tabelle *stationen* auswählen, erleben Sie eine Überraschung:

```
mysql> SELECT * FROM stationen;
Empty set (0.00 sec)
```

Die Daten werden nicht angezeigt – die oben beschriebene Isolation wirkt also.

Starten Sie nun auch im neuen Fenster eine Transaktion:

```
START TRANSACTION;
```

Als Nächstes sollen zwei weitere Datensätze hinzugefügt werden:

```
INSERT INTO stationen (s_name, s_dauer) VALUES
("Würzburg", 60);
INSERT INTO stationen (s_name, s_dauer) VALUES
("München", 50);
```

Durch `SELECT` lässt sich bestätigen, dass die neuen Daten angekommen sind, während die Daten der anderen Transaktion ausgeblendet bleiben (die `auto_increment`-IDs werden dagegen über die Transaktionen hinweg korrekt vergeben):

```
mysql> SELECT * FROM stationen;
+------+----------+---------+
| s_id | s_name   | s_dauer |
+------+----------+---------+
|    4 | Würzburg |      60 |
|    5 | München  |      50 |
+------+----------+---------+
```

Schalten Sie wieder um ins erste Fenster. Hier werden nach wie vor nur die ersten drei Datensätze angezeigt. Bestätigen Sie sie nun, indem Sie Folgendes eingeben:

```
mysql> COMMIT;
```

Im zweiten Fenster sind daraufhin alle fünf Datensätze vorhanden:

```
mysql> SELECT * FROM stationen;
+------+-----------+---------+
| s_id | s_name    | s_dauer |
+------+-----------+---------+
|    1 | K÷ln      |       0 |
|    2 | Bonn      |      30 |
|    3 | Frankfurt |      80 |
|    4 | Würzburg  |      60 |
|    5 | München   |      50 |
+------+-----------+---------+
```

Angenommen, die beiden letzten Datensätze sind doch unerwünscht. In diesem Fall lassen sie sich durch ein Rollback im zweiten Fenster auf einfache Weise wieder entfernen:

```
ROLLBACK;
```

Eine letzte `SELECT`-Anweisung zeigt dann, dass der Zustand in beiden Fenstern identisch ist:

```
mysql> SELECT * FROM stationen;
+------+-----------+---------+
| s_id | s_name    | s_dauer |
+------+-----------+---------+
|    1 | Köln      |       0 |
|    2 | Bonn      |      30 |
|    3 | Frankfurt |      80 |
+------+-----------+---------+
```

Views

Views (Ansichten) sind gespeicherte Tabellenansichten, die eine einfache Möglichkeit bieten, auf häufig genutzte Auswahlabfragen – mit oder ohne Verknüpfungen – zuzugreifen. Eine View ähnelt in vielerlei Hinsicht einer Tabelle – man kann daraus auswählen und unter bestimmten Umständen auch Daten darin ändern. Allerdings bezieht die View ihre Daten ausschließlich aus vorhandenen Tabellen und ändert sich automatisch mit diesen.

In MySQL werden Views seit Version 5.0 (in Binär-Releases seit 5.0.1) unterstützt. Eine View wird mithilfe der Abfrage `CREATE VIEW` erstellt, die folgende Grundsyntax besitzt:

```
CREATE VIEW ViewName AS
SELECT ...;
```

Die `SELECT`-Abfrage, die in der View abgespeichert wird, kann im Grunde aus allen im vorigen Kapitel gezeigten Komponenten bestehen. Das folgende Beispiel definiert eine View für die Datenbank *reisebuero*, die die Hotels den (benannten) Städten zuordnet:

```
CREATE VIEW hotel_orte AS
SELECT ht_name AS Hotel, st_name AS Stadt
FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr;
```

Diese View ist nun dauerhaft mit den Hotels und ihren Städten verknüpft – sobald ein Hotel geändert, hinzugefügt oder entfernt wird, zeigt sich das auch in der View. Die Informationen aus der View selbst lassen sich wiederum mithilfe einer beliebigen Auswahlabfrage auslesen. Das folgende Beispiel zeigt die ersten fünf Zeilen:

```
mysql> SELECT * FROM hotel_orte LIMIT 0,5;
+------------------+----------+
| Hotel            | Stadt    |
+------------------+----------+
| Bergerhof        | Köln     |
| Hotel Colonia    | Köln     |
| Hotel de la Gare | Paris    |
| Hotel au Jardin  | Paris    |
| Otel Bahar       | Istanbul |
+------------------+----------+
```

Eine besondere Form von Views sind *Updatable Views* – in ihnen können Informationen nicht nur gelesen, sondern auch verändert werden; dies ändert natürlich auch die Daten in der zugrunde liegenden Tabelle. Eine View besitzt diese Fähigkeit nur dann, wenn alle enthaltenen Spalten aus einer einzelnen Tabelle stammen – oder aus einer anderen View, die ihrerseits auf nur einer Tabelle basiert. Im Übrigen darf die Abfrage, die der View zugrunde liegt, keine Aggregatfunktionen enthalten.

Das folgende Beispiel erstellt eine einfache View auf die Tabelle *rb_airports* mit weniger Informationen:

```
CREATE VIEW air1 AS
SELECT ap_name, ap_kuerzel, ap_stadt FROM rb_airports;
```

Die folgende Einfügeabfrage erstellt einen neuen Datensatz für Sabiha, den zweiten Flughafen von Istanbul:

```
INSERT INTO air1 VALUES
("Istanbul Sabiha", "SAW", 3);
```

Dass die Daten tatsächlich in die Originaltabelle übernommen wurden, zeigt eine Auswahlabfrage, die alle Datensätze von *rb_airports* mit der Stadt Nummer 3 ausgibt:

```
mysql> SELECT ap_name, ap_stadt FROM rb_airports WHERE ap_stadt=3;
+------------------+----------+
| ap_name          | ap_stadt |
+------------------+----------+
| Istanbul Atatürk |        3 |
| Istanbul Sabiha  |        3 |
+------------------+----------+
```

Auch Löschabfragen aus Updatable Views sind möglich; und aus Gründen der Konsistenz mit den Beispielen im nächsten Kapitel sollten Sie Istanbul Sabiha nun tatsächlich wieder löschen:

```
DELETE FROM air1 WHERE ap_name="Istanbul Sabiha";
```

Eine erneute Auswahlabfrage für Stadt 3 dürfte zeigen, dass es funktioniert hat.

Zum Ändern einer View wird eine `ALTER VIEW`-Anweisung verwendet. Ihre Syntax ist mit `CREATE VIEW` identisch, so dass die gesamte `SELECT`-Abfrage wiederholt (und variiert) werden muss. Hier ein Beispiel, das *air1* um den Beinamen der Flughäfen erweitert:

```
ALTER VIEW air1 AS
SELECT ap_name, ap_zusatz, ap_kuerzel, ap_stadt FROM rb_airports;
```

Innerhalb der Datenbank erscheinen Views in derselben Übersicht wie gewöhnliche Tabellen – probieren Sie es aus und geben Sie Folgendes ein, um die Liste aller Tabellen und Views in der Datenbank *reisebuero* zu erhalten:

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_reisebuero |
+----------------------+
| air1                 |
| hotel_orte           |
| rb_airlines          |
| rb_airports          |
| rb_buchungen         |
| rb_fluege            |
| rb_flugstrecken      |
| rb_hotels            |
| rb_kunden            |
| rb_kundenkontakte    |
| rb_laender           |
| rb_sehensw           |
| rb_staedte           |
+----------------------+
```

Mit einem einfachen `DESC` können Sie auch nicht ermitteln, ob es sich bei einem Eintrag um eine Tabelle oder um eine View handelt. Beispiel:

```
mysql> DESC hotel_orte;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| Hotel | varchar(60) | NO   |     |         |       |
| Stadt | varchar(40) | NO   |     |         |       |
+-------+-------------+------+-----+---------+-------+
```

Erst eine `SHOW CREATE TABLE`-Abfrage enthüllt die wahre Natur einer View (abgesehen davon könnten Sie hier auch `SHOW CREATE VIEW` schreiben, was bei einer Tabelle zu einer Fehlermeldung führt). Auch dafür sehen Sie hier ein Beispiel:

```
mysql> SHOW CREATE TABLE hotel_orte \G
*************************** 1. row ***************************
       View: hotel_orte
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost`
SQL SECURITY DEFINER VIEW `hotel_orte` AS select `rb_hotels`.`ht_
name` AS `Hotel`,`rb_staedte`.`st_name` AS `Stadt` from (`rb_hotels`
join `rb_staedte` on((`rb_hotels`.`ht_stadt` = `rb_staedte`.`st_nr`)))
```

Sie können Views mithilfe der Anweisung `DROP VIEW` *`ViewName`* löschen – was Sie nun auch tun sollten, um in der Datenbank *reisebuero* wieder aufzuräumen:

```
DROP VIEW hotel_orte;
DROP VIEW air1;
```

Prepared Statements

Der Einsatz von *Prepared Statements* geht noch einen Schritt weiter als Views. Es handelt sich dabei um benannte SQL-Abfragen, die jederzeit mittels `EXECUTE` ausgeführt werden können. Manche Programmierschnittstellen für Datenbanken implementieren diese Funktionalität selbst, unabhängig davon, ob die zugrunde liegende Datenbank sie beherrscht. Im nächsten Kapitel lernen Sie beispielsweise die Prepared-Statement-Funktionen der PHP-Datenbankschnittstelle PDO kennen.

MySQL selbst unterstützt Prepared Statements seit Version 4.1. Die grundlegende Syntax zum Erstellen eines Prepared Statements sieht folgendermaßen aus:

```
PREPARE StatementName FROM Abfrage
```

Das folgende kurze Beispiel definiert eine Abfrage, die fünf zufällig ausgewählte Hotels, die zugehörigen Städte sowie den jeweiligen Einzel- und Doppelzimmerpreis ausgibt:

```
mysql> PREPARE random_hotels FROM
    -> 'SELECT ht_name AS Hotel, ht_ezpreis AS Einzel,
    '>  ht_dzpreis AS Doppel, st_name AS Stadt
    '>  FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
    '>  ORDER BY RAND() LIMIT 0,5';
Query OK, 0 rows affected (0.08 sec)
Statement prepared
```

Wie Sie sehen, ist es notwendig, die gesamte im Prepared Statement zu speichernde Abfrage in Anführungszeichen zu setzen. Sollte die Abfrage selbst Anführungszeichen benötigen, können Sie den jeweils anderen Typ (doppelte beziehungsweise einfache Anführungszeichen) verwenden. Zwei `EXECUTE`-Aufrufe zeigen, dass die zufällige Auswahl funktioniert:

```
mysql> EXECUTE random_hotels;
+-----------------------+--------+--------+-----------+
| Hotel                 | Einzel | Doppel | Stadt     |
+-----------------------+--------+--------+-----------+
| Hotel de la Gare      |     85 |    150 | Paris     |
| Hotel Vieu Lyon       |     60 |    100 | Lyon      |
| Fiaker-Hotel          |     80 |    125 | Wien      |
| Hotel Bahar            |     50 |     80 | Istanbul  |
| The New Maitland Hotel |     60 |    100 | Edinburgh |
+-----------------------+--------+--------+-----------+

mysql> EXECUTE random_hotels;
+-------------------+--------+--------+----------+
| Hotel             | Einzel | Doppel | Stadt    |
+-------------------+--------+--------+----------+
| Hotel Colonia     |    105 |    195 | Köln     |
| Haus Alpenhof     |     90 |    170 | Zürich   |
| Hotel Apollon     |     70 |    120 | Athen    |
| Old Palace Hotel  |    130 |    240 | London   |
| Hotel Solidarnosc |     40 |     70 | Warschau |
+-------------------+--------+--------+----------+
5 rows in set (0.01 sec)
```

Prepared Statements können beliebig oft den Platzhalter ? enthalten, der bei EXECUTE mit konkreten Werten gefüllt werden muss. Hier als Beispiel eine abgewandelte Variante der vorigen Abfrage. Sie enthält nur den Doppelzimmerpreis, für den ein Maximalwert angegeben werden kann:

```
mysql> PREPARE random_hotel_choice FROM
    -> 'SELECT ht_name AS Hotel, ht_dzpreis AS Doppel,
    '>  st_name AS Stadt
    '>  FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
    '>  WHERE ht_dzpreis<?
    '>  ORDER BY RAND() LIMIT 0,5';
```

Zur Übergabe eines konkreten Werts muss eine SQL-Benutzervariable definiert werden. Das erledigt eine SET-Anweisung wie diese:

```
SET @preis=100;
```

Nun kann das Prepared Statement ausgeführt werden; die Variable wird mithilfe einer USING-Klausel übergeben:

```
mysql> EXECUTE random_hotel_choice USING @preis;
+-------------------+--------+----------+
| Hotel             | Doppel | Stadt    |
+-------------------+--------+----------+
| Hotel Central     |     90 | Prag     |
| Hotel Solidarnosc |     70 | Warschau |
| Hotel Bahar        |     80 | Istanbul |
+-------------------+--------+----------+
```

Da es nur drei Hotels gibt, in denen ein Doppelzimmer weniger als 100 Euro pro Nacht kostet, werden nur diese angezeigt; allerdings wechselt die Reihenfolge bei jedem Aufruf.

Wenn Sie ein Prepared Statement nicht mehr verwenden möchten, können Sie es mithilfe der Anweisung `DEALLOCATE PREPARE` *`StatementName`* entfernen, zum Beispiel:

```
DEALLOCATE PREPARE random_hotel_choice;
```

Abgesehen davon werden Prepared Statements mit dem Ende der aktuellen MySQL-Clientsitzung automatisch gelöscht.

Stored Procedures

Der größte Nachteil von Prepared Statements ist die soeben erwähnte beschränkte Lebensdauer – nach einer einzelnen Clientsitzung ist Schluss. Die in MySQL 5.0 neu eingeführten *Stored Procedures* werden dagegen permanent auf dem Datenbankserver gespeichert. Datenbanksysteme, die Stored Procedures schon länger unterstützen, verwenden dafür meist jeweils eine eigene, inkompatible Syntax, da es früher keinen SQL-Standard dafür gab. MySQL benutzt dagegen die SQL:2003-Syntax für Stored Procedures.

Genau wie Prepared Statements sind auch Stored Procedures benannte Abfolgen von SQL-Abfragen. Allerdings sind sie erheblich leistungsfähiger, beispielsweise können sie eine beliebige Anzahl benannter Ein- und/oder Ausgabeparameter besitzen.

Zum Testen von Stored Procedures soll wieder einmal eine separate Datenbank zum Einsatz kommen; sie enthält nur zwei kleine Tabellen, in denen Flugzeugtypen und ihre Hersteller gespeichert werden. Erstellen Sie mithilfe der folgenden Anweisungen zunächst die Datenbank und die Tabellenstruktur:

```
CREATE DATABASE flieger;

USE flieger;

CREATE TABLE hersteller (
  h_id INT AUTO_INCREMENT PRIMARY KEY,
  h_name VARCHAR(40)
);

CREATE TABLE flugzeuge (
  f_id INT AUTO_INCREMENT PRIMARY KEY,
  f_typ VARCHAR(40),
  f_herst INT
);
```

Um mit dem Experimentieren zu beginnen, wird zunächst einmal je ein Datensatz eingefügt:

```
INSERT INTO hersteller (h_name) VALUES ("Boeing");
INSERT INTO flugzeuge (f_typ, f_herst) VALUES ("737-800", 1);
```

Die erste Stored Procedure soll einfach jeden Flugzeugtyp mit dem zugehörigen Hersteller ausgeben. Dazu wird eine `CREATE PROCEDURE`-Anweisung verwendet, deren allgemeines Syntaxschema wie folgt aussieht:

```
CREATE PROCEDURE ProzedurName ([IN|OUT|INOUT ParamName Typ[, ...]])
BEGIN
  MySQL-Anweisung[en]
END
```

Ein Problem ergibt sich bei der Eingabe von Stored Procedures im `mysql`-Client: Da die MySQL-Anweisungen zwischen `BEGIN` und `END` mit einem Semikolon enden, nimmt der Client an, dass die Eingabe abgeschlossen ist. Deshalb müssen Sie vor der Eingabe den `DELIMITER` (Eingabeabschluss, siehe Kapitel 4) auf eine Zeichenfolge setzen, die in der Stored Procedure selbst nicht vorkommt – hier zum Beispiel `//`:

```
mysql> \d //
```

Nun kann die Prozedur eingegeben werden:

```
CREATE PROCEDURE zeige_flugzeuge ()
BEGIN
  SELECT h_name, f_typ
  FROM hersteller INNER JOIN flugzeuge ON h_id=f_herst;
END
//
```

Natürlich ist es für die weitere Arbeit einfacher, den Delimiter nun zunächst zurückzusetzen:

```
mysql> \d ;
```

Jetzt können Sie die Prozedur mithilfe einer `CALL`-Anweisung ausprobieren:

```
mysql> CALL zeige_flugzeuge ();
+--------+---------+
| h_name | f_typ   |
+--------+---------+
| Boeing | 737-800 |
+--------+---------+
```

Die Klammern hinter dem Prozedurnamen sind normalerweise für Parameter gedacht – Felder mit einem beliebigen SQL-Datentyp; die einfache Prozedur `zeige_flugzeuge()` benötigt allerdings keine. Jeder Parameter wird durch eines von drei Schlüsselwörtern charakterisiert: `IN` kennzeichnet reine Eingabeparameter, die der Prozedur einen Wert übergeben. `OUT` bezeichnet Ausgabeparameter, die einen Wert an die `CALL`-Umgebung zurückgeben. `INOUT` schließlich erledigt beides in einem.

Das nächste Beispiel dient der einfachen Eingabe eines neuen Herstellers. Der `IN`-Parameter ist dessen Name. Geben Sie die folgende Prozedur mitsamt Austausch des Delimiters ein:

```
\d //

CREATE PROCEDURE neu_hersteller (IN h VARCHAR(40))
BEGIN
```

```
   INSERT INTO hersteller (h_name) VALUES (h);
END
//

\d ;
```

Wie Sie sehen, wird der Parameter h innerhalb der Stored Procedure ohne Anführungszeichen verwendet. Andernfalls würde MySQL den Wert natürlich als Zeichenkette "h" interpretieren. Aufgrund des Datentyps müssen die Anführungszeichen aber in einer CALL-Anweisung für diese Prozedur stehen:

```
CALL neu_hersteller ("Airbus");
```

Das folgende, letzte Beispiel bietet einen kleinen Ausblick auf die unzähligen weiteren Möglichkeiten von Stored Procedures. Es ermöglicht die Eingabe eines Flugzeugs samt Hersteller, wobei die Prozedur selbst überprüft, ob der angegebene Hersteller bereits existiert, und ihn gegebenenfalls hinzufügt. Auch den Bezug auf die Hersteller-ID stellt sie selbst her. Geben Sie zunächst den Code mit den obligatorischen Delimiter-Änderungen ein:

```
\d //

CREATE PROCEDURE neu_flugzeug (IN h VARCHAR(40), IN f VARCHAR(40))
BEGIN
   SET @anzahl=(SELECT COUNT(*) FROM hersteller WHERE h_name=h);
   IF @anzahl=0 THEN
      INSERT INTO hersteller (h_name) VALUES (h);
   END IF;
   SET @hnr=(SELECT h_id FROM hersteller WHERE h_name=h);
   INSERT INTO flugzeuge (f_typ, f_herst) VALUES (f, @hnr);
END
//

\d ;
```

Hier besteht natürlich ein wenig Erklärungsbedarf. Die programmiertechnischen Aspekte von SQL werden in diesem Buch ansonsten nicht behandelt; wenn Sie sich einigermaßen mit PHP auskennen, dürfte es aber keine Verständnisschwierigkeiten geben.

Zunächst wird ermittelt, ob der eingegebene Hersteller h bereits existiert. Dazu wird eine SELECT-Abfrage mit der Aggregatfunktion COUNT() ausgeführt. Ihr Ergebnis wird der Variablen @anzahl zugewiesen; die Syntax der Variablendefinition mit SET wurde bereits im Abschnitt über Prepared Statements vorgestellt. Es folgt eine – ebenfalls eher aus Programmiersprachen bekannte – IF-Anweisung mit folgender Syntax:

```
IF Ausdruck THEN
   Anweisung[en]
END IF;
```

Die verschachtelten Anweisungen werden nur ausgeführt, wenn die Anweisung wahr ist – in diesem Fall also, falls der Hersteller noch nicht bekannt ist. Die verschachtelte `INSERT`-Anweisung dient dazu, einen erwiesenermaßen neuen Hersteller in die Tabelle *hersteller* einzufügen.

Hinter dem `IF`-Block folgt eine weitere Variablendefinition: In `@hnr` wird die anhand des Herstellernamens ermittelte ID gespeichert, damit sie zusammen mit dem neuen Flugzeugtyp in die Tabelle *flugzeuge* eingefügt werden kann. Das geschieht in der letzten Zeile vor dem `END`.

Nun ist es an der Zeit, die neue Prozedur auszuprobieren. Die beiden folgenden Aufrufe verwenden einen vorhandenen und einen neuen Hersteller:

```
CALL neu_flugzeug ("Airbus", "A380");
CALL neu_flugzeug ("Tupolew", "Tu-144");
```

Ein erneuter Aufruf der Prozedur `zeige_flugzeuge()` sollte danach ergeben, dass alles wie erwartet funktioniert hat:

```
mysql> CALL zeige_flugzeuge();
+---------+---------+
| h_name  | f_typ   |
+---------+---------+
| Boeing  | 737-800 |
| Airbus  | A380    |
| Tupolew | Tu-144  |
+---------+---------+
```

Wenn Sie eine Stored Procedure ändern möchten, wird `ALTER PROCEDURE` verwendet; die Syntax ist mit `CREATE PROCEDURE` identisch. Löschen können Sie eine Prozedur mit `DROP PROCEDURE` *`ProzedurName`*.

Um eine Liste aller bestehenden Prozeduren zu erhalten, können Sie Folgendes eingeben (hier in vertikaler Darstellung, da die Tabellenansicht zu breit wäre):

```
mysql> SHOW PROCEDURE STATUS \G
*************************** 1. row ***************************
           Db: flieger
         Name: neu_flugzeug
         Type: PROCEDURE
      Definer: root@localhost
     Modified: 2007-03-05 11:07:51
      Created: 2007-03-05 11:07:51
Security_type: DEFINER
      Comment:
*************************** 2. row ***************************
           Db: flieger
         Name: neu_hersteller
         Type: PROCEDURE
      Definer: root@localhost
     Modified: 2007-03-05 11:07:20
      Created: 2007-03-05 11:07:20
Security_type: DEFINER
      Comment:
```

```
*************************** 3. row ***************************
           Db: flieger
         Name: zeige_flugzeuge
         Type: PROCEDURE
      Definer: root@localhost
     Modified: 2007-03-05 11:06:34
      Created: 2007-03-05 11:06:34
Security_type: DEFINER
      Comment:
```

Eine besondere Form der Stored Procedures sind *Stored Functions*. Diese werden mittels `CREATE FUNCTION` erstellt und geben einen Wert zurück. Der Datentyp dieses Werts muss mithilfe einer `RETURNS`-Klausel angegeben werden; die eigentliche Wertrückgabe erfolgt mittels `RETURN`-Anweisung. Stored Functions können nicht auf Datensätze aus Tabellen zugreifen, sondern nur auf ihre Eingabeparameter und daraus berechnete Werte.

Hier ein ganz einfaches Beispiel, das die beiden numerischen Eingabewerte addiert und die Summe zurückgibt:

```
CREATE FUNCTION summe (z1 INT, z2 INT) RETURNS INT
BEGIN
   RETURN z1 + z2;
END
```

Eine weitere Besonderheit: Da in Stored Functions ausschließlich Eingabeparameter erlaubt sind, wird die Parameterart nicht durch ein Schlüsselwort gekennzeichnet.

Der Aufruf einer solchen Funktion erfolgt nicht durch `CALL`, sondern im Kontext eines beliebigen Ausdrucks – vorausgesetzt, der Datentyp passt zur Umgebung. Hier als Beispiel eine `SELECT`-Abfrage, die `summe()` verwendet:

```
mysql> SELECT summe(10, 20);
+---------------+
| summe(10, 20) |
+---------------+
|            30 |
+---------------+
```

Trigger

Ein *Trigger* ist eine SQL-Anweisung oder eine Abfolge von Anweisungen, die automatisch vor oder nach einer bestimmten Datenänderungsabfrage aufgerufen wird. Somit ermöglichen Trigger die Ausführung zusätzlicher Arbeitsschritte nach dem Einfügen, Ändern oder Löschen von Daten. Sinnvolle Anwendungen sind beispielsweise Plausibilitätskontrollen für eingefügte Werte oder eine komplexere Bearbeitung von Fremdschlüsselbeziehungen. In MySQL sind Trigger seit Version 5.0 verfügbar; erst im Release 5.0.10 wurden sie so weit ergänzt, dass sich sinnvoll damit arbeiten lässt.

Die allgemeine Syntax für einen Trigger lautet:

```
CREATE TRIGGER Triggername BEFORE|AFTER INSERT|UPDATE|DELETE
On Tabellenname FOR EACH ROW SQL-Anweisung(en)
```

Den Triggernamen können Sie frei wählen; er muss nur innerhalb der jeweiligen Datenbank eindeutig sein. `BEFORE` oder `AFTER` bestimmt, ob der Trigger unmittelbar vor oder unmittelbar nach der mit ihm verknüpften Operation aufgerufen wird. Anschließend wird die Art der Abfragen festgelegt, für die der Trigger gilt; die möglichen Schlüsselwörter `INSERT`, `UPDATE` und `DELETE` kennen Sie bereits aus den vorigen Kapiteln. Der Tabellenname gibt natürlich an, für welche Tabelle der angegebene Abfragetyp überwacht wird. Hinter `FOR EACH ROW` folgt eine oder mehrere SQL-Anweisungen. Wenn Sie mehrere verwenden möchten, müssen Sie wie bei Stored Procedures `BEGIN` und `END` einsetzen; auch die Änderung des Delimiters ist in diesem Fall wieder erforderlich.

In den SQL-Anweisungen des Triggers können Sie Bezug auf die Spalten der Tabelle nehmen, die von der untersuchten Abfrage betroffen ist. Dazu stehen die speziellen Formulierungen `OLD.`*`Feldname`* und `NEW.`*`Feldname`* zur Verfügung; dabei steht `OLD` für den Inhalt vor der Änderung und `NEW` für den geänderten Inhalt. Bei `INSERT` ist nur `NEW` verfügbar, bei `DELETE` nur `OLD`; bei `UPDATE` schließlich können Sie beide Varianten einsetzen.

Als Beispiel sehen Sie hier einen Trigger, der bei der Änderung des Doppelzimmerpreises eines Hotels überprüft, ob dieser immer noch höher als der eines Einzelzimmers ist. Falls dies nicht mehr der Fall ist, wird er automatisch auf den Einzelzimmerpreis plus 10 Euro gesetzt:

```
\d //

CREATE TRIGGER ht_dzpreis_test BEFORE UPDATE
ON rb_hotels FOR EACH ROW
BEGIN
  IF new.ht_dzpreis <= new.ht_ezpreis THEN
    SET new.ht_dzpreis=new.ht_ezpreis + 10;
  END IF;
END

//

\d ;
```

Speichern Sie die Preise eines beliebigen Hotels zunächst in Variablen, bevor Sie ein Änderungsexperiment durchführen. Hier als Beispiel das Hotel Nummer 10:

```
mysql> SET @ezpreis =
    -> (SELECT ht_ezpreis FROM rb_hotels WHERE ht_nr=10);

mysql> SET @dzpreis =
    -> (SELECT ht_dzpreis FROM rb_hotels WHERE ht_nr=10);
```

Schauen Sie sich die bisherigen Preise an:

```
mysql> SELECT @ezpreis, @dzpreis;
+----------+----------+
| @ezpreis | @dzpreis |
+----------+----------+
| 100      | 185      |
+----------+----------+
```

Ändern Sie nun einfach den Einzelzimmpreis dieses Hotels:

```
mysql> UPDATE rb_hotels SET ht_ezpreis=190 WHERE ht_nr=10;
```

Das Auslesen der Daten zeigt, dass der Trigger funktioniert hat:

```
mysql> SELECT ht_name, ht_ezpreis, ht_dzpreis FROM rb_hotels
    -> WHERE ht_nr=10;
+----------+------------+------------+
| ht_name  | ht_ezpreis | ht_dzpreis |
+----------+------------+------------+
| La Barca |        190 |        200 |
+----------+------------+------------+
```

Danach können Sie die Werte wieder zurücksetzen und die Werte erneut überprüfen:

```
mysql> UPDATE rb_hotels SET ht_ezpreis=@ezpreis, ht_dzpreis=@dzpreis
    -> WHERE ht_nr=10;

mysql> SELECT ht_name, ht_ezpreis, ht_dzpreis FROM rb_hotels
    -> WHERE ht_nr=10;
+----------+------------+------------+
| ht_name  | ht_ezpreis | ht_dzpreis |
+----------+------------+------------+
| La Barca |        100 |        185 |
+----------+------------+------------+
```

Um künftig wieder vollkommen unabhängig über die Hotelpreise bestimmen zu können, sollten Sie den Trigger nun wieder löschen:

```
DROP TRIGGER ht_dzpreis_test;
```

Eine Übersicht über die vorhandenen Trigger der aktuellen Datenbank erhalten Sie übrigens mithilfe der folgenden Anweisung:

```
SHOW TRIGGERS;
```

In diesem Kapitel:

- PHP-Grundlagen
- Die MySQL-Schnittstellen in PHP
- Clientseitiges Scripting mit JavaScript und Ajax
- Die Reisebüro-Anwendung

KAPITEL 8

Webanwendungen mit PHP und MySQL

Erst durch des Wissens Verwendung
erfüllt sich des Weisen Sendung.
Jüdisches Sprichwort

Das vorliegende Kapitel ist in gewisser Weise der Kern dieses Buchs: Hier erfahren Sie, wie das Datenbanksystem MySQL und die Programmiersprache PHP 5 miteinander verknüpft werden, um leistungsfähige Webanwendungen zu programmieren.

Im ersten Abschnitt werden einige PHP-Grundlagen vermittelt; dafür sind, wie bereits erwähnt, nur einige HTML-Vorkenntnisse nötig. Anschließend erhalten Sie einen systematischen Überblick über die bereits angesprochenen MySQL-Schnittstellen von PHP, *mysql*, *mysqli* und *PHP Data Objects*. Im letzten Abschnitt schließlich wird die MySQL-basierte PHP-Webanwendung des Reisebüros vorgestellt. Dabei werden einige der Skripten vollständig abgedruckt und erläutert, andere dagegen aus Platzgründen eher kurz gestreift.

Die vollständige Reisebüro-Site finden Sie auf der beiliegenden CD-ROM. Es lohnt sich, sie zu installieren, auszuprobieren und auch die Quellcodes der restlichen Skripten zu studieren. Sie enthalten zahlreiche erläuternde Kommentare.

PHP-Grundlagen

In diesem Abschnitt werden einige grundlegende Konzepte der Programmiersprache PHP vorgestellt, die in den Anwendungsbeispielen am Ende dieses Kapitels ohne weitere Erläuterung vorausgesetzt werden. Der erste der beiden Unterabschnitte behandelt Allgemeines zur Sprache, der zweite webspezifische Funktionen.

Falls Sie bereits mit PHP vertraut sind, können Sie diesen Abschnitt komplett überspringen. Benötigen Sie dagegen eine gründlichere Einführung in die PHP-Programmierung, kann ich Ihnen das Buch *PHP 5 – Ein praktischer Einstieg* von Ulrich Günther empfehlen, das ebenfalls in der Buchreihe *O'Reillys Basics* erschienen ist.

Allgemeine Sprachmerkmale

Wie Sie wahrscheinlich bereits wissen – oder in den Einführungsbeispielen in den Kapiteln 2 und 3 erfahren haben –, ist eine PHP-Datei formal ein gewöhnliches HTML-Dokument, in dem bestimmte Blöcke wie folgt als ausführbare PHP-Anweisungen gekennzeichnet werden:

```
<?php ... ?>
```

Bevor der Webserver das Dokument an den Browser eines Besuchers ausliefert, werden diese Blöcke vom PHP-Interpreter ausgeführt. Dabei werden die Ausgabetexte von `echo()`-Anweisungen an den entsprechenden Stellen in den HTML-Code eingefügt. Auf diese Weise macht PHP es Ihnen leicht, statische und dynamische Bereiche in einem Dokument beliebig zu mischen.

Innerhalb der PHP-Bereiche stehen PHP-Anweisungen, deren Syntax an klassische Sprachen wie C oder auch Perl erinnert. Die beiden neuesten Versionen, PHP 4 und 5, wurden allerdings stark um objektorientierte Fähigkeiten erweitert; ihre Funktionsweise wurde weitgehend von Java inspiriert.

Variablen

Variablen sind in PHP leicht an dem führenden Dollarzeichen (`$`) zu erkennen. Das erste Zeichen hinter diesem Präfix muss ein Buchstabe oder Unterstrich sein; danach können Buchstaben, Ziffern oder Unterstriche folgen. Groß- und Kleinschreibung werden unterschieden. Wie die meisten Skriptsprachen ist PHP untypisiert – eine Variable kann nacheinander Werte verschiedener Datentypen annehmen. Hier ein Beispiel für unterschiedliche Wertzuweisungen:

```
$var = "Hallo";
$var = 7;       // legal, keine Fehlermeldung
```

Unter Umständen werden Variablenwerte je nach Kontext unterschiedlich interpretiert, zum Beispiel:

```
$a = "42";      // String durch Anführungszeichen
$a += 23;       // numerische Addition -> Gesamtwert 65
$a .= "56";     // String-Verkettung -> Ergebnis "6556"
```

Übrigens existieren Variablen automatisch ab der ersten Wertzuweisung; eine Deklaration ist nicht nötig.

Eine interessante Besonderheit von PHP ist die automatische *Variablensubstitution* in Strings – Variablen werden anhand des Dollarzeichens erkannt und automatisch ausgewertet. Hier ein Beispiel:

```
$betrag = 100;
echo ("Ich habe $betrag Euro.<br />");
```

Das HTML-Ergebnis sieht folgendermaßen aus:

```
Ich habe 100 Euro.<br />
```

Wenn Sie in einer Zeichenkette ein Dollarzeichen benötigen, müssen Sie eine Escape-Sequenz daraus machen, indem Sie ihm einen Backslash voranstellen, zum Beispiel:

```
echo ("Ich habe $betrag \$.<br />");
// Ergebnis: Ich habe 100 $.<br />
```

Übrigens können Sie die Substitution unterdrücken, indem Sie einfache Anführungszeichen verwenden. Das obige Beispiel in dieser Schreibweise führt daher zu einem anderen Ergebnis:

```
echo ('Ich habe $betrag \$.<br />');
// Ergebnis: Ich habe $betrag \$.<br />
```

Operatoren

In jeder Programmiersprache gibt es unterschiedliche Operatoren zur Konstruktion komplexer Ausdrücke. Hier eine kurze Übersicht der wichtigsten PHP-Operatoren:

- Die arithmetischen Operatoren sind mit denjenigen von MySQL identisch, die im vorigen Kapitel vorgestellt wurden. Ausnahme: Der Modulo-Operator (Rest der ganzzahligen Division) wird ausschließlich als Prozentzeichen (%) geschrieben.
- Der Zuweisungsoperator = wurde bereits erwähnt. Er dient dazu, einer Variablen oder Eigenschaft einen beliebigen Ausdruck als Wert zuzuweisen, zum Beispiel:

  ```
  $text = "So long, and thanks for all the fish.";
  ```
- Eine weitere Gruppe von Operatoren erleichtert die Modifikation vorhandener Variablen durch eine abkürzende Schreibweise. Beispielsweise können Sie statt

  ```
  $a = $a + 2;
  ```

  Folgendes schreiben:

  ```
  $a += 2;
  ```

  Dasselbe gilt für alle arithmetischen und noch einige weitere Operatoren.
- Für das Erhöhen (*Inkrement*) oder Vermindern (*Dekrement*) um 1 gibt es sogar noch eine kompaktere Form. Die folgenden Anweisungen weisen $a den Wert 4 zu und erhöhen den Wert der Variablen dann um 1:

  ```
  $a = 4;
  $a++;
  ```

  Es gibt einen kleinen, aber wichtigen Unterschied zwischen einem vorangestellten ++, dem sogenannten *Prä-Inkrement*, und der nachgestellten Version, dem *Post-Inkrement*: Innerhalb eines komplexeren Ausdrucks wird beim Prä-Inkrement der neue, beim Post-Inkrement dagegen der alte Wert der betreffenden Variablen verwendet. Beispiele:

```
$a = 3;
// Post-Inkrement:
$b = $a++;  // $a ist nun 4, $b ist 3
// Prä-Inkrement:
$c = ++$a;  // $a und $c haben nun den Wert 5
```

Dasselbe gilt übrigens für die Verminderung um 1 mittels --, die entsprechend als Prä- beziehungsweise Post-Dekrement bezeichnet wird.

- Als Vergleichsoperatoren sind == (gleich), != (ungleich), <, >, <= und >= verfügbar. Anders als in vielen anderen Programmiersprachen können sie sowohl für Zahlen als auch für Strings verwendet werden. Letzteres richtet sich streng nach der Zeichensatzabfolge – Sie können die Regeln nachvollziehen, wenn Sie die Informationen über Binary-Kollationen im vorigen Kapitel lesen.
- Die logischen Operatoren && (logisches Und), || (logisches Oder) und ! (logisches Nicht) funktionieren ebenfalls genau wie in MySQL.
- Der im Beispiel weiter oben verwendete Punkt (.) dient der Verkettung von Strings. Allerdings ist er aufgrund der Variablensubstitution oft überflüssig.

Kontrollstrukturen

Mit zu den wichtigsten Bestandteilen von Programmiersprachen zählen die sogenannten *Kontrollstrukturen*. Sie ermöglichen die bedingte und gegebenenfalls auch wiederholte Ausführung von Anweisungen.

Die häufigste Kontrollstruktur ist die einfache *Fallentscheidung* mit if, die folgende grundlegende Syntax besitzt:

```
if (Ausdruck) {
   Anweisung[en];
}
```

Der Anweisungsblock wird dabei nur ausgeführt, wenn der zu prüfende Ausdruck wahr ist. Das folgende Beispiel gibt "Köln liegt in Deutschland.
" aus, wenn $stadt den Wert "Köln" hat:

```
if ($stadt == "Köln") {
   echo ("$stadt liegt in Deutschland.<br />");
}
```

Optional können Sie einen else-Block hinzufügen, dessen Inhalt ausgeführt wird, falls die Bedingung nicht zutrifft, zum Beispiel:

```
if ($stadt == "Köln") {
   echo ("Die Stadt ist K&ouml;ln.<br />");
} else {
   echo ("Die Stadt ist nicht K&ouml;ln.<br />");
}
```

Schließlich existiert mit `elseif` noch eine Anweisung für verschachtelte Bedingungen, die nur überprüft werden, falls die vorherige `if`-Bedingung nicht zutraf. Auch dafür sehen Sie hier ein Beispiel:

```
if ($stadt == "Köln") {
   echo ("Die Stadt ist K&ouml;ln.<br />");
} elseif ($stadt == "Paris") {
   echo ("Die Stadt ist Paris.<br />");
} else {
   echo ("Die Stadt ist weder K&ouml;ln noch Paris.<br />");
}
```

Eine weitere Form von Kontrollstrukturen sind die *Schleifen*. Sie führen eine Folge von Anweisungen mehrmals aus, wobei auch sie in der Regel eine Bedingung prüfen. Die einfachste Form ist die `while`-Schleife; ihre Syntax lautet folgendermaßen:

```
while (Ausdruck) {
   Anweisung[en];
}
```

Solange der Ausdruck wahr ist, werden die Anweisungen immer wieder ausgeführt. Das folgende Beispiel gibt die Quadrate der Zahlen 1 bis 10 aus:

```
// Startwert: $i = 1
$i = 1;
// Hat $i höchstens den Wert 10?
while ($i <= 10) {
   $quadrat = $i * $i;
   echo ("$i<sup>2</sup> = $quadrat<br />");
   // $i um 1 erhöhen
   $i++;
}
```

Da die Bedingung bereits vor der ersten Ausführung des Schleifenrumpfs, das heißt der abhängigen Anweisungen, überprüft wird, wird dieser Schleifentyp als *kopfgesteuerte Schleife* bezeichnet. Dabei kann es vorkommen, dass der Schleifenrumpf gar nicht ausgeführt wird. Ein anderer Typ, die sogenannte *fußgesteuerte Schleife*, überprüft die Bedingung dagegen nach dem ersten Durchlauf. Hier sehen Sie das obige Beispiel in dieser Schreibweise:

```
// Startwert: $i = 1
$i = 1;
do {
   $quadrat = $i * $i;
   echo ("$i<sup>2</sup> = $quadrat<br />");
   // $i um 1 erhöhen
   $i++;
} while ($i <= 10);
```

Neben der `while`-Schleife gibt es in PHP auch die `for`-Schleife. Sie hat folgende Syntax:

```
for (Initialisierung; Bedingung; Wertänderung) {
   Anweisung[en];
}
```

Die *Initialisierung* ist eine beliebige Anweisung, die einmal vor dem ersten Schleifendurchlauf ausgeführt wird. Die *Bedingung* ist ein beliebiger Ausdruck; der Schleifenrumpf wird nur (erneut) ausgeführt, wenn dieser wahr ist. Die *Wertänderung* schließlich wird nach jedem Durchlauf ausgeführt. Hier sehen Sie das Quadratzahlen-Beispiel zum dritten Mal, diesmal als `for`-Schleife:

```
for ($i = 1; $i <= 10; $i++) {
   $quadrat = $i * $i;
   echo ("$i<sup>2</sup> = $quadrat<br />");
}
```

`for` ist also lediglich eine kompaktere Schreibweise für kopfgesteuerte Schleifen, insbesondere für solche mit einer festgelegten (determinierten) Anzahl von Abläufen.

Arrays

Arrays sind spezielle Variablen, in denen mehrere Werte listenartig gespeichert werden können. Das eröffnet zahlreiche Automatisierungsmöglichkeiten. Jedes Element in einem Array besitzt einen Index, über den es identifiziert werden kann; dieser steht in eckigen Klammern hinter dem Variablennamen. In PHP kann der Index je nach Wunsch numerisch oder ein String sein – anders als zum Beispiel in Perl, wo String-Indizes den sogenannten Hashes, einem anderen Variablentyp, vorbehalten sind.

Ein Array wird entweder durch Wertzuweisung an ein einzelnes Element oder durch die Funktion `array()` erstellt. Beispiele:

```
// Einzelne Elemente
$laender [0] = "Deutschland";
$laender [1] = "Frankreich";
$laender [2] = "Türkei";

// Funktion array():
$staedte = array ("Köln", "Paris", "Istanbul");
```

Numerische Arrays werden, wie Sie sehen, ab 0 durchnummeriert, im zweiten Beispiel automatisch. `$staedte [0]` hat also den Wert `"Köln"`.

String-Indizes werden in Anführungszeichen in die eckigen Klammern geschrieben; in der Funktion `array()` dient der spezielle Operator `=>` der Verknüpfung von Index und Element. Auch dazu sehen Sie hier wieder zwei Beispiele:

```
// Einzelne Elemente
$airlines ['LH'] = "Lufthansa";
$airlines ['4U'] = "Germanwings";
$airlines ['AF'] = "Air France";

// Funktion array():
$airports = array (
   'CGN' => "Köln/Bonn",
   'IST' => "Istanbul",
   'FCO' => "Roma Fiumicino");
```

Wie bereits in früheren Kapiteln erwähnt wurde, werden auch MySQL-Datensätze als Arrays ausgelesen. Sie können sich sogar aussuchen, ob sie durchnummeriert werden oder ob die Feldnamen die Indizes bilden sollen.

Um alle Elemente eines numerischen Arrays auszugeben, können Sie ein Schleifenkonstrukt wie dieses verwenden:

```
$anzahl = sizeof ($laender);
for ($i = 0; $i < $anzahl; $i++) {
   echo ($laender [$i]."<br />");
}
```

Die Ausgabe dieses Beispiels im Browser sieht folgendermaßen aus:

Deutschland

Frankreich

Türkei

Mithilfe der Funktion `sizeof($array)` wird die Anzahl der Elemente im Array ermittelt.

Für die Ausgabe aller Elemente eines Arrays mit String-Indizes bietet sich dagegen die Funktion `each($array)` an, die bei jedem Aufruf das nächste Paar aus Schlüssel und Wert zurückgibt. Mithilfe der bereits in früheren Kapiteln erwähnten Funktion `list()` können Sie dieses Paar einer Liste aus zwei Variablen zuweisen. Der Code zur Ausgabe der Fluggesellschaften und ihrer Kürzel sieht also wie folgt aus:

```
// Array-Zähler auf das Anfangselement setzen:
reset ($airlines);
while (list ($index, $wert) = each ($airlines)) {
   echo ("$wert ($index)<br />");
}
```

Wenn Sie dieses Beispiel ausführen, zeigt der Browser Folgendes an:

Köln/Bonn (CGN)

Istanbul (IST)

Roma Fiumicino (FCO)

Mithilfe einfacher Funktionen können Sie Arrays übrigens zu Strings zusammenfassen und umgekehrt Strings zerlegen. `explode ($muster, $string)` zerlegt `$string` an jeder Stelle, an der `$muster` vorkommt, zum Beispiel:

```
$airportlist = "Köln/Bonn, London-Heathrow, Roma Fiumicino";
$airports = explode (", ", $string);
/* $airports [0] ist "Köln/Bonn"
   $airports [1] ist "London-Heathrow"
   $airports [2] ist "Roma Fiumicino"  */
```

Die Funktion `implode($muster, $array)` erledigt das Gegenteil – sie fasst `$array` zu einem String zusammen, in dem die bisherigen Elemente durch `$muster` getrennt werden. Dazu dieses Beispiel:

```
$airlines = array ("Turkish Airlines", "KLM", "Iberia");
$airlinelist = implode (", ", $airlines);
// $airlinelist ist "Turkish Airlines, KLM, Iberia"
```

Reguläre Ausdrücke

Genau wie MySQL unterstützt auch PHP reguläre Ausdrücke, und zwar sogar in mehreren Implementierungen. Die leistungsfähigste Variante sind die Funktionen für Perl-kompatible reguläre Ausdrücke (PCRE). Die Grundlagen der RegExp-Syntax selbst können Sie in Kapitel 6 nachschlagen. Beachten Sie, dass `[:Klasse:]`-Konstrukte in PCRE nicht zur Verfügung stehen.

Mithilfe der Funktion `preg_match ($regexp, $string)` können Sie überprüfen, ob der reguläre Ausdruck `$regexp` auf `$string` passt. Das folgende Beispiel überprüft, ob der Inhalt der Variablen `$name` mit *A* beginnt:

```
if (preg_match ("/^A/", $name) {
   echo ("$name beginnt mit A<br />");
} else {
   echo ("$name beginnt nicht mit A<br />");
}
```

Der reguläre Ausdruck muss in Anführungszeichen sowie zwischen Slashes (`//`) stehen. Hinter dem zweiten Slash können Modifikatoren folgen. Beispielsweise sorgt `i` dafür, dass nicht zwischen Groß- und Kleinschreibung unterschieden wird. Sie können zum Beispiel folgendermaßen nach allen Varianten von »mysql« suchen:

```
if (preg_match ("/mysql/i", $text)) {
   echo ("Gl&uuml;ckwunsch, MySQL vorhanden.<br />");
}
```

Mit `preg_replace ($regexp, $ersatz, $string[, $limit])` wird der reguläre Ausdruck `$regexp` in `$string` durch den Ersatztext `$ersatz` ersetzt. Standardmäßig ersetzt die Funktion jedes Vorkommen von `$regexp`; mithilfe des optionalen Parameters `$limit` können Sie die Anzahl der Ersetzungen allerdings beschränken. Das folgende Beispiel ersetzt in `$text` alle Vorkommen von »Lufthansa« durch »Germanwings«:

```
$text = preg_replace ("/Lufthansa/i", "Germanwings", $text);
```

Im Ersatztext können geklammerte Teilausdrücke aus dem regulären Ausdruck wieder eingefügt werden; sie werden dabei als `$1` bis `$99` durchnummeriert. Das folgende Beispiel kehrt einen Namen nach dem Schema »Vorname Nachname« (zum Beispiel »David Axmark«) in »Nachname, Vorname« (»Axmark, David«) um:

```
$name = preg_replace ("/([a-z]+)\s+([a-z]+)/", "$2, $1", $name);
```

Funktionen

In Computerprogrammen kommt es oft vor, dass dieselbe Abfolge von Anweisungen mehrmals benötigt wird. In solchen Fällen lohnt es sich, diese Anweisungen in eine *Funktion* auszulagern. Dabei handelt es sich um einen benannten Anweisungsblock mit folgender Syntax:

```
function FunktionsName ([Param1[=Standard], Param2[=Standard], ...]) {
   Anweisung[en];
   [return Ausdruck;]
}
```

Der Aufruf einer solchen Funktion führt die enthaltenen Anweisungen aus; eine eventuell vorhandene `return`-Anweisung gibt einen Wert an die aufrufende Stelle zurück. In den Klammern können die Namen von Parametervariablen stehen. Beim Aufruf müssen diese als konkrete Werte angegeben werden; innerhalb der Funktion stehen sie dann unter dem entsprechenden Variablennamen zur Verfügung. Das folgende Beispiel definiert eine Funktion namens `gruss()`, die die angegebene Person je nach Tageszeit unterschiedlich begrüßt:

```
function gruss ($name="Anonymous") {
   // Stunde ermitteln
   $jetzt = time();
   $stunde = date ("G", $jetzt);
   // Grußwort je nach Stunde
   if ($stunde < 12) {
      echo ("Guten Morgen, ");
   } elseif ($stunde < 18) {
      echo ("Guten Tag, ");
   } else {
      echo ("Guten Abend, ");
   }
   // den Namen ausgeben
   echo ("$name!<br />");
}
```

Die PHP-Funktion `time()` gibt die Sekunden seit EPOCH zurück. `date($format, $time)` formatiert eine solche Angabe ähnlich wie die in Kapitel 6 vorgestellte MySQL-Funktion `DATE_FORMAT()` – allerdings steht das Format hier vor der Zeit, und die Platzhalter beginnen nicht mit einem Prozentzeichen. `G` steht für die Stundenangabe im 24-Stunden-Modus ohne führende Null bei einstelligen Angaben. Eine Liste aller zulässigen `date()`-Formate finden Sie im Verzeichnis *docs* auf der beiliegenden CD-ROM.

Üblicherweise sollten Funktionen in einem PHP-Block zu Beginn des Dokuments gespeichert werden.

Ein Aufruf von `gruss()` sieht beispielsweise so aus:

```
gruss ("Klaus");
```

Um 13:23 Uhr erhalten Sie folgendes Ergebnis:

Guten Tag, Klaus!

Da für `$name` der Standardwert `"Anonymous"` angegeben wurde, kann die Funktion alternativ auch ohne Argument aufgerufen werden:

```
gruss();
```

Dies liefert um 9:53 Uhr beispielsweise diese Ausgabe:

Guten Morgen, Anonymous!

Funktionen, die einen Wert zurückgeben, können innerhalb von Ausdrücken aufgerufen werden. Das nächste kleine Beispiel zeigt die Funktion `ist_gerade()`, die überprüft, ob die übergebene Zahl gerade oder ungerade ist, und dementsprechend 1 beziehungsweise 0 zurückgibt:

```
function ist_gerade ($zahl) {
   // Keine Zahl? Natürlich nicht gerade!
   if (!is_numeric ($zahl)) {
      return 0;
   }
   // Gerade, falls durch 2 teilbar
   if ($zahl % 2 == 0) {
      return 1;
   }
   // Noch hier? Dann ungerade!
   return 0;
}
```

Wie Sie sehen, besitzt `ist_gerade()` insgesamt drei `return`-Anweisungen, von denen zwei von Fallentscheidungen abhängen. Auffällig ist, dass keiner der beiden `if`-Blöcke ein `else` besitzt. Das ist möglich, weil jedes `return` die Funktion sofort verlässt und mit dem angegebenen Wert zur aufrufenden Stelle zurückkehrt.

Die PHP-Funktion `is_numeric()` überprüft, ob das Argument eine Zahl ist. PHP kennt mehrere solcher Testfunktionen. Hier nur einige Beispiele: `is_int()` testet auf Ganzzahlen, `is_float()` auf Fließkommazahlen und `is_string()` auf Zeichenketten, und `is_null()` überprüft, ob das Argument `null` (eine leere Referenz) ist. Der eigentliche Test, ob die Zahl gerade ist, wird mithilfe des Modulo-Operators durchgeführt: Wenn die Division durch 2 den Rest 0 ergibt, ist die Zahl durch 2 teilbar und damit gerade.

Die selbst definierte Funktion `ist_gerade()` kann nun beispielsweise in einer `if`-Fallentscheidung verwendet werden:

```
if (ist_gerade ($zahl)) {
   echo ("$zahl ist eine gerade Zahl.<br />");
} else {
   echo ("$zahl ist keine gerade Zahl.<br />");
}
```

Objektorientierung

Die *objektorientierte Programmierung* (OOP) ist eine in den 1970er-Jahren eingeführte Programmiertechnik. Es geht im Wesentlichen darum, Datenstrukturen fest mit den zu ihrer Verarbeitung vorgesehenen Funktionen zu verknüpfen; dieses Konzept wird *Kapselung* genannt. In der traditionellen *imperativen* Programmierung greift eine Funktion von außen auf die Datenstruktur zu, während die objektorientierte Datenstruktur selbst »weiß«, was sie tun kann – sie enthält neben den Daten gekapselte Funktionen, die *Methoden* genannt werden. Das hat mehrere Vorteile:

- Das Verhalten von Objekten aus der realen Welt lässt sich auf diese Weise realistischer in Software modellieren. Beispielsweise bestünde die imperative Abbildung eines Flugzeugs aus einer Ansammlung von Zustandsdaten wie Tankfüllung, Kilometerstand und aktuellem Standort, die durch Funktionen von außen modifiziert würden. Das objektorientierte Flugzeug hätte dagegen Methoden wie `fliegen()` oder `tanken()`, um diese Daten selbst zu ändern.
- Die Kapselung führt außerdem zu weniger Fehlern: Da die Daten nur noch durch die Methoden der Objekte selbst manipuliert werden, werden Programme viel übersichtlicher – der fehlerträchtige Einsatz globaler, an vielerlei Stellen modifizierter Datenstrukturen entfällt.
- Hauptbestandteil objektorientierter Programme sind die sogenannten *Klassen*; sie bilden die Vorlagen oder »Bauanleitungen« für Objekte. Die einzelnen Klassen lassen sich viel leichter wiederverwenden als die Datenstrukturen und Funktionen imperativer Programme; es ist einfach und praktisch, sich nach und nach eigene Klassenbibliotheken zu erstellen.
- Ein weiterer Vorteil der Klassen ist die *Vererbung*: Jede Klasse kann von einer anderen abgeleitet werden, so dass jeweils nur die Besonderheiten und Unterschiede neu programmiert werden müssen. Das spart Schreibarbeit und verbessert ebenfalls die Wiederverwendbarkeit.

Seit Version 4 verfügt auch PHP über objektorientierte Merkmale, die in PHP 5 noch einmal stark erweitert wurden. Syntax und Funktionsweise wurden an Java angelehnt.

Um objektorientiert zu programmieren, wird zunächst eine Klasse definiert. Das geschieht mit dem Schlüsselwort `class`. Genau wie Funktionen sollten auch Klassendefinitionen in einem PHP-Block am Dokumentbeginn stehen. Hier ein einfaches Beispiel, das eine Klasse für die Erstellung von HTML-Formularen definiert:

```
class Form {

   // Konstruktor
   function __construct ($u, $m) {
      // Eigenschaften initialisieren
      $this->url = $u;
      $this->method = $m;
   }
```

```
    // Methoden
    function start () {
       echo ("<form action=\"{$this->url}\"
              method=\"{$this->method}\">\n");
    }

    function textfield ($n) {
       echo ("<input type=\"text\" name=\"$n\" width=\"40\" />
              <br />\n");
    }

    function radiogroup ($n, $werte, $texte) {
       for ($i = 0; $i < sizeof($werte); $i++) {
          echo ("<input type=\"radio\" name=\"$n\"
                value=\"{$werte [$i]}\" />{$texte [$i]}<br />\n");
       }
    }

    function submit ($v) {
       echo ("<input type=\"submit\" value=\"$v\" /><br />\n");
    }

    function end () {
       echo ("</form>\n");
    }
}
```

Es wurde bereits erwähnt, dass eine Klasse lediglich eine Vorlage für Objekte ist. Mithilfe des Operators `new` wird ein Objekt, eine sogenannte *Instanz* der Klasse, erstellt. Dabei wird automatisch der Konstruktor aufgerufen, eine Funktion namens `__construct()` mit *zwei* führenden Unterstrichen.[1] In der Regel wird der Konstruktor dazu genutzt, um den *Eigenschaften* – also den Datenstrukturvariablen – der Klasse ihre Anfangswerte zuzuweisen. In PHP werden diese durch das Präfix `$this` gekennzeichnet.

Das folgende Beispiel erstellt eine Formular-Instanz mit der Action-URL *test.php* und der HTTP-Methode `GET`:

```
$form = new Form ("test.php", "GET");
```

Neben dem Konstruktor verfügt die Klasse über fünf Methoden, die jeweils durch `$instanz->`*methode*`()` aufgerufen werden. Hier ein Beispiel für mehrere Formularinhalte:

```
// <form ...> ausgeben:
$form->start ();
echo ("Ihr Name: ");
// Textfeld
$form->textfield ("user");
echo ("Wohin m&ouml;chten Sie reisen?<br />\n");
```

1 In PHP 4 ist der Konstruktor eine Funktion, die den Namen der Klasse selbst trägt.

```
// Parameter für die Radio-Buttons:
$werte = array ("CGN", "FCO", "IST");
$namen = array ("K&ouml;ln", "Rom", "Istanbul");
// Radio-Button-Gruppe:
$form->radiogroup ("ziel", $werte, $namen);
// Absendeknopf
$form->submit ("Abschicken");
// </form> ausgeben:
$form->end ();
```

Im Browser eines Besuchers entsteht durch die Formular-Methoden der folgende HTML-Code:

```
<form action="test.php" method="GET">
Ihr Name: <input type="text" name="user" width="40" /><br />
Wohin m&ouml;chten Sie reisen?<br />
<input type="radio" name="ziel" value="CGN" />K&ouml;ln<br />
<input type="radio" name="ziel" value="FCO" />Rom<br />
<input type="radio" name="ziel" value="IST" />Istanbul<br />
<input type="submit" value="Abschicken" /><br />
</form>
```

Natürlich hat die Klasse in dieser Form noch keinen besonderen Nutzen – aber es wäre zum Beispiel ohne Weiteres denkbar, sie um Formatierungen mittels Tabellen oder CSS zu erweitern.

Web-Features

Dieser kurze Abschnitt behandelt zwei wichtige Techniken, die in größeren Webanwendungen häufig benötigt werden: Sessions und Cookies. Die bedeutendste webspezifische Funktion – das Einlesen von Formulardaten – wurde bereits in Kapitel 3 ausführlich behandelt.

Sessions

Das Webprotokoll HTTP ist *zustandslos* – zwischen zwei Anfragen desselben Browsers an denselben Server besteht keine Verbindung. Bei der Durchführung von Geschäftsabschlüssen und sonstigen umfangreicheren Transaktionen über das Web ist das ein Problem, da sich alle Aktionen desselben Besuchers aufeinander beziehen müssen, etwa um beliebig viele Artikel in einen virtuellen Warenkorb zu legen und später alle auf einmal zu bestellen.

Eine häufig genutzte Lösung für dieses Problem ist das sogenannte *Session-Tracking*. Dabei wird beim Aufruf der ersten Seite durch einen bestimmten Benutzer eine eindeutige Nummer generiert, die Session-ID. Diese wird an die URL jedes Hyperlinks und jeder Formulardatenübertragung angehängt, so dass jedes einzelne Skript der Webanwendung erkennen kann, zu welchem Besucher der Aufruf gehört. So können Daten zwischen den einzelnen Seiten weitergereicht werden, beispielsweise über eine Datenbank.

PHP bietet ein sehr einfaches Session-Verfahren: Die Session-ID wird automatisch im Hintergrund generiert und weitergereicht; in dem globalen Array `$_SESSION` können Sie beliebig viele Werte ablegen, die für jeden einzelnen Besucher zwischen den PHP-Skripten hin- und hergereicht werden. Technisch verwendet PHP zur Weitergabe der Session-Daten Cookies (siehe unten), wenn der Browser sie akzeptiert. Andernfalls wird die Session-ID automatisch an die URLs angehängt, während die Session-Daten in einer temporären Datei auf dem Server abgelegt werden.

Jede Seite, die Teil einer Session sein soll, benötigt als Erstes folgende PHP-Anweisung:

```
session_start();
```

Diese muss in einem PHP-Block ganz am Anfang der Datei stehen; vor dem öffnenden `<?php` darf nicht einmal ein einziges Leerzeichen stehen. Nach `session_start()` können Sie beliebige Elemente aus `$_SESSION` auslesen und neue hineinschreiben. Hier ein kurzes Gesamtbeispiel:

```
<?php

   // Session-Zugriff vorbereiten
   session_start();
   // Alten Session-Wert Reiseziel lesen
   $reiseziel = $_SESSION['ziel'];

   // ... Formulardaten auslesen, z.B. Reisedatum ...

   // Reisedatum in die Session schreiben:
   $_SESSION['datum'] = $reisedatum;

?>
```



Die weiter unten ausführlich vorgestellte Reisebüro-Anwendung verwendet Sessions, um die Benutzeran- und -abmeldung zu verwalten.

Cookies

Cookies lösen ein ähnliches Problem wie Sessions, allerdings können sie Daten auch für eine spätere Sitzung aufbewahren. Sie werden nämlich nicht auf dem Server gespeichert, sondern durch den Browser auf dem Clientrechner abgelegt (für die konkrete Implementierung haben die Browserhersteller verschiedene Methoden gewählt). Greift der Browser später wieder auf denselben Server zu, sendet er dessen Cookies automatisch mit jeder Anfrage.

Das Problem ist, dass Sie sich nicht bei allen Besuchern auf die Verfügbarkeit von Cookies verlassen können: Viele Benutzer deaktivieren sie komplett, weil sie nicht damit einverstanden sind, dass viele Onlinewerber Cookies dazu missbrauchen, ihr Surfverhalten auszuspionieren.

In PHP werden Cookies über die Funktion `setcookie()` gesetzt, die folgende Syntax besitzt:

```
setcookie ($name, $wert[, $verfallsdatum[, $pfad
     [, $domain[, $secure]]]])
```

Die verschiedenen Parameter bedeuten Folgendes:

- `$name`: Name des Cookies, über den Sie es beim späteren Abruf identifizieren können.
- `$wert`: Die konkrete Information, die in dem Cookie gespeichert werden soll.
- `$verfallsdatum`: Ende der Gültigkeit des Cookies in Sekunden seit EPOCH. Addieren Sie einfach `time()` und die gewünschte Sekundenzahl – zum Beispiel 86.400 für einen ganzen Tag. Cookies ohne Verfallsdatum sind Session-Cookies, die nur innerhalb der aktuellen Clientsitzung gelten.
- `$pfad`: Übergeordneter URL-Pfad der Dokumente, die das Cookie lesen dürfen.
- `$domain`: Domain, unter der das Cookie gilt.
- `$secure`: 1 oder 0. 1 bedeutet, dass das Cookie nur über eine gesicherte SSL-Verbindung übertragen wird.

Da Cookies als HTTP-Header gesetzt werden, muss `setcookie()` – genau wie die Session-Anweisungen – in einem PHP-Block am Anfang der Datei verwendet werden. Das folgende Beispiel setzt ein 14 Tage lang gültiges Cookie mit dem Namen `letzterbesuch`, dessen Wert der aktuelle Zeitpunkt – also `time()` – ist:

```
setcookie ("letzterbesuch", time(), time() + 14 * 86400);
```

Um Cookies wieder auszulesen, steht das globale Array `$_COOKIE` zur Verfügung. Hier ein Beispiel, das das Cookie `letzterbesuch` wieder ausliest:

```
$besuch = $_COOKIE ['lastvisit'];
$besuchformat = date ("d.m.Y, H:i", $besuch);
echo ("Ihr letzter Besuch war am $besuchformat.<br />");
```

Die MySQL-Schnittstellen in PHP

Die drei MySQL-Schnittstellen der Programmiersprache PHP – das klassische *mysql*, das moderne, objektorientierte *mysqli* und die Datenbankabstraktion *PHP Data Objects* (PDO) – wurden in diesem Buch bereits intuitiv verwendet. In diesem Abschnitt werden sie systematisch vorgestellt.

mysql und mysqli

Tabelle 8-1 zeigt zunächst einen Überblick über die wichtigsten Elemente der beiden Schnittstellen *mysql* und *mysqli*. Beachten Sie, dass *mysqli* hier nur in objektorientierter Syntax dargestellt wird; der ebenfalls verfügbare, weitgehend an *mysql* angelehnte imperative Zugriff wird in diesem Buch nicht behandelt.

Tabelle 8-1: Vergleich der wichtigsten Elemente der Schnittstellen mysql und mysqli

Funktionalität	mysql	mysqli
Verbindungsaufbau	`$id = mysql_connect ($host,$user,$pass);`	`$conn = news mysqli ($host,$user,$pass,$db);`
Datenbankauswahl	`mysql_select_db ($db);`	`$conn->select_db ($db);`
Abfrage	`$query = mysql_query ($sql);`	`$query = $conn->query ($sql);`
Datensatz lesen (numerisches Array)	`$arr = mysql_fetch_ row ($query);`	`$arr = $query->fetch_row ();`
Datensatz lesen (benanntes Array)	`$arr = mysql_fetch_ array ($query);`	`$arr = $query->fetch_array ();`
Anzahl Ergebnisdaten-sätze (Auswahlabfrage)	`$lines = mysql_num_ rows($query);`	`$lines = $query->num_rows;`
Anzahl geänderter Zei-len (Änderungsabfragen)	`$lines = mysql_affected_ rows();`	`$lines = $conn->affected_rows`
Zuletzt eingefügte auto_ increment-ID	`$id = mysql_insert_id();`	`$id = $conn->insert_id;`
Verbindung schließen	`mysql_close ();`	`$conn->close ();`

Der erste Schritt besteht jeweils darin, eine Verbindung zum Datenbankserver herzustellen. Die Parameter sind Hostname, Benutzername und Passwort. Bei mysqli kommt zusätzlich die Standarddatenbank hinzu. Wenn Sie dem Buch bis hier gefolgt sind, besitzen Sie noch keinen separaten Benutzer für die Datenbank *reisebuero*. Starten Sie deshalb den Kommandozeilenclient mysql und legen Sie diesen Benutzer wie folgt an:

```
mysql> GRANT ALL PRIVILEGES ON reisebuero.* TO rbuser@localhost
    -> IDENTIFIED BY 'R3153n';
mysql> FLUSH PRIVILEGES;
```

Denken Sie wieder daran, dass Sie ein anderes Passwort verwenden sollten als das hier im Beispiel gezeigte.

Nun kann dieser neue Benutzer in einer PHP-Testanwendung eine Verbindung zum Datenbankserver herstellen. In der *mysql*-Version sieht das so aus:

```
$connID = mysql_connect ("localhost", "rbuser", "R3153n");
```

Die Verbindungsnummer `$connID` brauchen Sie nur zu benutzen, wenn mehrere parallele Datenbankverbindungen bestehen – für diesen Fall gibt es alternative Fassungen der *mysql*-Funktionen, die als zusätzlichen (letzten) Parameter eine solche ID entgegennehmen.

Bei *mysql* muss als Nächstes *reisebuero* als Standarddatenbank ausgewählt werden:

```
mysql_select_db ("reisebuero");
```

Die modernere Schnittstelle erledigt beide Schritte gleichzeitig im `new`-Aufruf der *mysqli*-Instanz:

```
$conn = new mysqli ("localhost", "rbuser", "R3153n", "reisebuero");
```

Dennoch verfügt auch *mysqli* über eine Methode zur Auswahl der Standarddatenbank, damit Sie diese nachträglich wechseln können. Das folgende Beispiel wählt die Datenbank *gewinnspiel* aus Kapitel 3 als Standard aus:

```
$conn->select_db ("gewinnspiel");
```

An die bestehende Verbindung können nun beliebige SQL-Abfragen übermittelt werden. Als Beispiel soll die folgende Abfrage dienen, die alle Hotels mitsamt Einzel- und Doppelzimmerpreisen und den Namen der Städte ausgibt, sortiert nach Städten und sekundär nach Hotelnamen:

```
SELECT ht_name, ht_ezpreis, ht_dzpreis, st_name
FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
ORDER BY st_name ASC, ht_name ASC
```

In *mysql*-Syntax wird die Abfrage wie folgt an den Server übermittelt:

```
// Abfrageinhalt
$querytext = "SELECT ht_name, ht_ezpreis, ht_dzpreis, st_name
             FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
             ORDER BY st_name ASC, ht_name ASC";
// Abfrage an Server senden
$query = mysql_query ($querytext);
```

mysqli verlangt dagegen diese Schreibweise:

```
// Abfrage an Server senden
$query = $conn->query ($querytext);
```

Nach dem Absenden einer Auswahlabfrage sollten Sie in der Regel zunächst überprüfen, ob es überhaupt Ergebnisdatensätze gibt. Dafür ist bei der klassischen Schnittstelle die Funktion `mysql_num_rows()` zuständig:

```
if (mysql_num_rows ($query) > 0) {
  // Ergebnisdatensätze verarbeiten
} else {
  // Z.B. Ausgabe, dass keine Daten vorhanden sind
}
```

Bei *mysqli* ist die Eigenschaft `num_rows` des jeweiligen Query-Objekts dafür zuständig. Da es sich um eine Eigenschaft und nicht um eine Methode handelt, dürfen auch keine Parameterklammern dahinter gesetzt werden:

```
if ($query->num_rows > 0) {
  ...
}
```

Sobald bekannt ist, dass ein Abfrageergebnis vorhanden ist, können dessen Datensätze nacheinander ausgelesen werden. Erfreulicherweise lassen sich die Funktionen `mysql[i]_fetch_row()` beziehungsweise `mysql[i]_fetch_array()` einfach in der Bedingung einer `while`-Schleife verwenden, so dass die Schleife automatisch für jede Ergebniszeile ausgeführt wird.

Hier zunächst die *mysql*-Fassung mit `mysql_fetch_row()`; diese Funktion liefert den Datensatz als durchnummeriertes Array zurück. Die überaus praktische PHP-Funktion `list()` bündelt eine Gruppe von Variablen, um ihnen nacheinander die Elemente eines Arrays als Werte zuzuweisen:

```
while (list ($hotel, $ezpreis, $dzpreis, $stadt) =
             mysql_fetch_row ($query)) {
  echo ("<tr>\n");
  echo ("<td>$hotel</td>\n");
  echo ("<td align=\"right\">$ezpreis &euro;</td>\n");
  echo ("<td align=\"right\">$dzpreis &euro;</td>\n");
  echo ("<td>$stadt</td>\n");
  echo ("</tr>\n");
}
```

Wie Sie sehen, werden die einzelnen Feldinhalte als Zeile einer (weiter oben im Dokument gestarteten) HTML-Tabelle ausgegeben.

Auch *mysqli* besitzt eine `fetch_row()`-Methode; die entsprechende Zeile (die `while`-Bedingung) sieht hier wie folgt aus:

```
while (list ($hotel, $ezpreis, $dzpreis, $stadt) =
       $query->fetch_row()) {
  // ... Ausgabe siehe oben ...
}
```

`mysql_fetch_array()` liefert den Datensatz ebenfalls als Array, allerdings sind die Indizes hier die Spaltennamen aus der Abfrage. Damit könnten Sie die Schleife auch folgendermaßen schreiben:

```
while ($record = mysql_fetch_array ($result)) {
  echo ("<tr>\n");
  echo ("<td>".$record ['ht_name']."</td>\n");
  echo ("<td align=\"right\">".$record ['ht_ezpreis'].
        " &euro;</td>\n");
  echo ("<td align=\"right\">".$record ['ht_dzpreis'].
        " &euro;</td>\n");
  echo ("<td>".$record ['st_name']."</td>\n");
  echo ("</tr>\n");
}
```

Bei *mysqli* liegt der einzige Unterschied wieder in der `while`-Zeile:

```
while ($record = $query->fetch_array()) {
   ...
}
```



Wenn Sie ausgelesenen Spalten einen Aliasnamen zuweisen, dient dieser als Index des entsprechenden Ergebnisfelds. Bei berechneten Ergebnisfeldern ist dies sogar die einzige Möglichkeit, `fetch_array()` zu verwenden. Das folgende Beispiel liest die Namen der Kölner Hotels und ihre Netto-Einzelzimmerpreise mittels *mysqli* aus und gibt sie zeilenweise aus.

```
$sql = "SELECT ht_name hotel, round(ht_ezpreis/1.
19, 2) netto
        FROM rb_hotels WHERE ht_stadt=1";
$query = $conn->query ($sql);
while ($row = $query->fetch_array()) {
  echo ($row['hotel'].": ".$row['netto'].
" &euro; netto<br />");
}
```

Nach getaner Arbeit kann die Datenbankverbindung geschlossen werden. Hier die *mysql*-Version:

```
mysql_close ();
```

In *mysqli* sieht die Anweisung wie folgt aus:

```
$conn->close ();
```

Beispiel 8-1 zeigt das gesamte Hotellistenskript in *mysql*-Syntax.

Beispiel 8-1: Ausgabe einer Liste aller Hotels, mysql-Version

```
<html>
  <head>
    <title>Reiseb&uuml;ro: Hotels</title>
  </head>
  <body>
    <h1>Liste aller Hotels</h1>
    <table border="2" cellpadding="4">
      <tr>
        <th>Hotel</th>
        <th>Einzelzimmer</th>
        <th>Doppelzimmer</th>
        <th>Stadt</th>
      </tr>
<?php

   // Datenbankparameter
   $host = "localhost";
   $user = "rbuser";
   $pass = "R3153n";
   $db = "reisebuero";

   // Verbindung herstellen
   $connID = mysql_connect ($host, $user, $pass);

   // Datenbank auswählen
   mysql_select_db ($db);

   // Abfrageinhalt
   $querytext = "SELECT ht_name, ht_ezpreis, ht_dzpreis, st_name
                 FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
```

Beispiel 8-1: Ausgabe einer Liste aller Hotels, mysql-Version (Fortsetzung)

```
                 ORDER BY st_name ASC, ht_name ASC";
   // Abfrage an Server senden
   $result = mysql_query ($querytext);

   // Ausgabeschleife
   while (list ($hotel, $ezpreis, $dzpreis, $stadt) = mysql_fetch_row ($result)) {
      echo ("<tr>\n");
      echo ("<td>$hotel</td>\n");
      echo ("<td align=\"right\">$ezpreis &euro;</td>\n");
      echo ("<td align=\"right\">$dzpreis &euro;</td>\n");
      echo ("<td>$stadt</td>\n");
      echo ("</tr>\n");
   }

   // Datenbankverbindung schließen
   mysql_close();

?>
    </table>
  </body>
</html>
```

In Beispiel 8-2 sehen Sie dasselbe kurze Skript noch einmal in der *mysqli*-Variante.

Beispiel 8-2: Ausgabe einer Liste aller Hotels, mysqli-Version

```
<html>
  <head>
    <title>Reiseb&uuml;ro: Hotels</title>
  </head>
  <body>
    <h1>Liste aller Hotels</h1>
    <table border="2" cellpadding="4">
      <tr>
        <th>Hotel</th>
        <th>Einzelzimmer</th>
        <th>Doppelzimmer</th>
        <th>Stadt</th>
      </tr>
<?php

   // Datenbankparameter
   $host = "localhost";
   $user = "rbuser";
   $pass = "R3153n";
   $db = "reisebuero";

   // Verbindung herstellen, Datenbank wählen
   $conn = new mysqli ($host, $user, $pass, $db);

   // Abfrageinhalt
```

Beispiel 8-2: Ausgabe einer Liste aller Hotels, mysqli-Version (Fortsetzung)

```
   $querytext = "SELECT ht_name, ht_ezpreis, ht_dzpreis, st_name
                 FROM rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
                 ORDER BY st_name ASC, ht_name ASC";
   // Abfrage an Server senden
   $query = $conn->query ($querytext);

   // Ausgabeschleife
   while (list ($hotel, $ezpreis, $dzpreis, $stadt) = $query->fetch_row ()) {
      echo ("<tr>\n");
      echo ("<td>$hotel</td>\n");
      echo ("<td align=\"right\">$ezpreis &euro;</td>\n");
      echo ("<td align=\"right\">$dzpreis &euro;</td>\n");
      echo ("<td>$stadt</td>\n");
      echo ("</tr>\n");
   }

   // Datenbankverbindung schließen
   $conn->close();

?>
    </table>
  </body>
</html>
```

Änderungsabfragen werden ebenfalls mit `mysql_query()` beziehungsweise `mysqli->query()` an den Server gesendet. Da diese Abfragen keine Datensätze zurückgeben, müssen Erfolg oder Misserfolg anders festgestellt werden: Die Funktion `mysql_affected_rows()` beziehungsweise die *mysqli*-Eigenschaft `affected_rows` geben jeweils die Anzahl der bei der letzten Operation geänderten Datensätze an.

Es folgt ein kurzer Beispielausschnitt, der das Prinzip demonstriert: Das *Hotel Konigin Juliana* in Amsterdam soll in *Hotel Konigin Beatrix* umbenannt werden. Die Abfrage kann als erfolgreich betrachtet werden, wenn genau ein Datensatz geändert wurde. Hier die *mysql*-Version:

```
// Inhalt der Abfrage
$querytext = "UPDATE rb_hotels SET ht_name=\"Hotel Konigin Beatrix\"
              WHERE ht_name LIKE \"%Juliana%\"";
// Abfrage an Server senden
mysql_query ($querytext);
// Hat es funktioniert?
if (mysql_affected_rows () == 1) {
  echo ("&Auml;nderung erfolgt.<br />\n");
} else {
  echo ("&Auml;nderung nicht m&ouml;glich.<br />\n");
}
```

In *mysqli* ist `affected_rows` keine Methode, sondern eine Eigenschaft – das zeigt sich wieder an den fehlenden Parameterklammern. Hier dasselbe Beispiel in dieser Variante:

```
// Inhalt der Abfrage
$querytext = "UPDATE rb_hotels SET ht_name=\"Hotel Konigin Beatrix\"
              WHERE ht_name LIKE \"%Juliana%\"";
// Abfrage an Server senden
$conn->query ($querytext);
// Hat es funktioniert?
if ($conn->affected_rows == 1) {
  echo ("&Auml;nderung erfolgt.<br />\n");
} else {
  echo ("&Auml;nderung nicht m&ouml;glich.<br />\n");
}
```

PHP Data Objects (PDO)

Die beiden oben vorgestellten Schnittstellen *mysql* und *mysqli* dienen exklusiv dem Zugriff auf MySQL-Datenbanken. Mit *PHP Data Objects* (kurz PDO) wurde dagegen eine neue Möglichkeit eingeführt, die auch in anderen Programmiersprachen immer häufiger vorkommt: eine sogenannte Abstraktion der Datenbankverbindungen. PDO ermöglicht also über jeweils angepasste Treiber die Nutzung verschiedener konkreter Datenbanken mithilfe einer gemeinsamen Syntax. Dies macht es für Entwickler leichter, die einer Anwendung zugrunde liegende Datenbanksoftware nachträglich auszutauschen oder die Entscheidung über die einzusetzende Datenbank dem Endanwender zu überlassen.

Beachten Sie, dass es andere Datenbank-Abstraktionsschichten gibt, die auch die Abfragen selbst abstrahieren, also keine SQL-Strings mehr an die Datenbank senden, sondern eigene Methoden dafür anbieten. So weit geht PDO nicht – wenn Sie die Datenbank wechseln und ihr SQL-Dialekt sich von der vorherigen unterscheidet, müssen Sie den SQL-Code selbst anpassen; zur Unterstützung mehrerer Datenbanken sind eventuell `if`-`elseif`-`else`-Konstrukte erforderlich.

In Tabelle 8-2 sehen Sie zunächst eine Schnellübersicht über die wichtigsten PDO-Konstrukte, analog zu der weiter oben gezeigten Tabelle für *mysql* und *mysqli*.

Tabelle 8-2: Übersicht der wichtigsten PDO-Elemente

Funktionalität	PDO-Code
Verbindungsaufbau	`$conn = new PDO ("mysql:host=$host;dbname=$db", $user, $pass[, $option_array]);`
Datenbankauswahl	(nicht vorgesehen; bei Bedarf neue Verbindung herstellen)
Abfrage	`$query = $conn->query ($sql);`
Datensatz lesen (numerisches Array)	`$arr = $query->fetch (PDO::FETCH_NUM);`
Datensatz lesen (benanntes Array)	`$arr = $query->fetch (PDO::FETCH_ASSOC);`

Tabelle 8-2: Übersicht der wichtigsten PDO-Elemente (Fortsetzung)

Funktionalität	PDO-Code
Anzahl Ergebnisdatensätze (Auswahlabfrage)	(nicht verfügbar; wenn Sie die Anzahl vorab benötigen, ist eine separate `SELECT COUNT(...)`-Abfrage erforderlich)
Anzahl geänderter Zeilen (Änderungsabfragen)	`$lines = $query->rowCount();`
Zuletzt eingefügte auto_increment-ID	`$id = $conn->lastInsertId();`
Verbindung schließen	`$conn = null;`

Die PDO-Verbindung und ihre Optionen

Zum Aufbau einer PDO-Verbindung wird eine neue PDO-Instanz erzeugt. Der Konstruktoraufruf hat folgende allgemeine Syntax:

```
$conn = new PDO ($datenbank_url, $user, $passwort[, $option_array]);
```

Das erste Argument, ein String mit der Datenbank-URL, sieht je nach verwendetem Datenbanktreiber unterschiedlich aus, hat aber im Allgemeinen folgendes Format:

```
"Treiber:Schlüssel=Wert[;Schlüssel=Wert[;...]]"
```

Der Treiber für MySQL heißt `mysql`; die beiden wichtigsten Schlüssel sind `host` für den Hostnamen und `dbname` für den Namen der Standarddatenbank. Das folgende Beispiel ist die URL für eine Verbindung zur MySQL-Datenbank *reisebuero* auf dem lokalen Rechner:

```
"mysql:host=localhost;dbname=reisebuero"
```

Die nächsten beiden Argumente sind ebenfalls Strings; sie enthalten den Datenbank-Benutzernamen und das zugehörige Passwort und funktionieren wie bei den anderen Schnittstellen. Sie sind nicht für alle Datenbanktreiber nötig, aber für MySQL durchaus.

Eine PDO-Besonderheit ist das optionale letzte Element: Sie können ein assoziatives Array aus `Schlüssel=>Wert`-Paaren hinzufügen, das diverse Optionen für die Verbindung festlegt. Die Schlüssel sind dabei Konstanten der Klassen PDO im Format `PDO ::KONSTANTE`. Die möglichen Werte vieler Konstanten sind wahr oder falsch, wofür typischerweise 1 und 0 verwendet werden. Die wichtigsten sind:

`PDO::ATTR_PERSISTENT => 1|0`
: Wenn Sie diesen Wert auf 1 setzen, wird versucht, eine persistente (dauerhafte) Verbindung wiederzuverwenden. Dies beschleunigt stark frequentierte Anwendungen und schont ihren Ressourcenverbrauch. Den Wert 0 brauchen Sie nicht explizit zu setzen, da er Standard ist.

`PDO::ATTR_AUTOCOMMIT => 1|0`
: Wenn die angesprochenen Tabellen Transaktionen unterstützen (bei MySQL also InnoDB-Tabellen), wird normalerweise standardmäßig der Autocommit-Modus verwendet. Weisen Sie diesem Schlüssel dagegen den Wert 0 zu, wird er deaktiviert, so dass die neue Verbindung eine Transaktion beginnt.

`PDO::ATTR_ERRMODE =>`
`PDO::ERRMODE_SILENT|PDO::ERRMODE_WARNING|PDO::ERRMODE_EXCEPTION`

Der Schlüssel `PDO::ATTR_ERRMODE` bestimmt, wie PDO auf Fehler in der SQL-Syntax, in Tabellen- und Spaltennamen oder anderen Aspekten Ihrer Abfragen reagiert.

Der Standardwert `PDO::ERRMODE_SILENT` hat das übliche Verhalten bei Fehlern zur Folge: Der Fehler wird beim Absenden der Abfrage nicht gemeldet, sondern das resultierende Ergebnisobjekt (hier `PDOStatement`) ist ungültig.

Wenn Sie den Wert `PDO::ERRMODE_WARNING` wählen, wird bei Abfragefehlern eine `E_WARNING`-Warnmeldung generiert. Damit solche Meldungen ausgegeben werden, müssen Sie zu Beginn Ihres Skripts folgende Zeile einfügen:

```
error_reporting (E_ERROR | E_WARNING);
```

Der Wert `PDO::ERRMODE_EXCEPTION` schließlich sorgt dafür, dass bei Abfragefehlern eine PDOException ausgelöst wird. Diese können Sie mittels `catch()` abfangen, wenn Sie alle Ihre PDO-Operationen in einen `try{}`-Block verschachteln. Beispiel:

```
try {
  // Datenbankverbindung mit Persistenz und Exception-Modus
  $conn = new PDO ("mysql:host=localhost;dbname=reisebuero",
                  $user, $host, array (
                   PDO::ATTR_PERSISTENT => 1,
                   PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
  // fehlerhafte Abfrage (falscher Tabellenname) zum Testen
  $query = $conn->query ("SELECT * FROM rb_hotel");
} catch (PDOException $e) {
  echo ("Fehler: ".$e->getMessage());
}
```

Wenn Sie dieses Skript ausführen, erhalten Sie folgende Fehlermeldung: »Fehler: SQLSTATE[42S02]: Base table or view not found: 1146 Table 'reisebuero.rb_hotel' doesn't exist«



Während der Entwicklungsphase ist diese Art von Fehlermeldung sehr nützlich. Den Endanwender sollten Sie allerdings nicht damit behelligen. Falls Sie in einer öffentlich verfügbaren Webanwendung Exceptions abfangen, ist es deshalb ratsam, benutzerfreundlichere Fehlermeldungen zu schreiben. Dabei können Sie sich aber durchaus an den Fehlernummern oder SQLSTATE-Codes orientieren; in Anhang D finden Sie die URL einer entsprechenden Liste.

Übrigens gibt es auch eine Möglichkeit, PDO-Attribute nach dem Verbindungsaufbau zu setzen oder zu ändern. Dafür ist die Methode `setAttribute ($attribut, $wert)` zuständig. Das folgende Beispiel stellt nachträglich den Fehlermodus `PDO::ERRMODE_EXCEPTION` ein:

```
$conn->setAttribute (PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Wenn Sie alle Datenbankoperationen in Ihrem Skript abgeschlossen haben, sollten Sie die Datenbankverbindung schließen. PDO kennt keine `close()`-Methode oder dergleichen; es genügt, der bisherigen Instanz den Wert `null` zuzuweisen:

```
$conn = null;
```

Abfragen mit PDO durchführen

Um eine SQL-Abfrage zu senden, wird die PDO-Methode `query()` verwendet. Genau wie die gleichnamige *mysqli*-Methode nimmt sie als Argument einen String mit SQL-Code entgegen. Sie liefert ein `PDOStatement`-Objekt zurück, aus dem Sie bei Auswahlabfragen die Ergebnisdatensätze oder bei Änderungsabfragen die Anzahl der betroffenen Datensätze auslesen können.

Eine Besonderheit der Methode `query()` ist, dass sie anstelle eines `fetch()`-Aufrufs auch selbst als Schleifenbedingung fungieren kann, um das Ergebnis einer Auswahlabfrage zeilenweise auszulesen. Das folgende Beispiel geht davon aus, dass `$conn` eine gültige PDO-Verbindung zur Datenbank *reisebuero* ist, und liest zeilenweise die Namen und Städte der Hotels aus:

```
$sql = "SELECT ht_name, st_name FROM rb_hotels INNER JOIN rb_staedte
        ON ht_stadt=st_nr ORDER BY st_name, ht_name";
foreach ($conn->query($sql) as $row) {
  echo ($row['ht_name'].", ".$row['st_name']."<br />\n");
}
```

Bei dieser Herangehensweise stehen die Ergebnisdatensätze sowohl mit numerischen Indizes als auch mit Spaltennamenschlüsseln zur Verfügung. Statt `$row['ht_name']` und `$row['st_name']` könnten Sie also auch `$row[0]` beziehungsweise `$row[1]` schreiben. Auch wenn Sie `fetch()` als Schleifenbedingung verwenden, ist dies standardmäßig der Fall. Allerdings können Sie dann optional eine bestimmte Array-Variante wählen, indem Sie `fetch()` eine der folgenden Konstanten als Parameter übergeben (es gibt weitere Varianten, die hier aus Platzgründen wegfallen):

- `PDO::FETCH_NUM` liefert die Ergebnisdatensätze nur als nummerierte Arrays.
- `PDO::FETCH_NAMED|PDO::FETCH_ASSOC` liefert nur die benannten Schlüssel.
- `PDO::FETCH_BOTH` ist Standard; beide Varianten werden zurückgeliefert.
- `PDO::FETCH_LAZY` ist eine PDO-Besonderheit: Jede Ergebniszeile ist ein Objekt; die Felder stehen als Instanzvariablen mit den Namen der Spalten zur Verfügung.

Hier eine Variante des obigen Beispiels, die explizit nummerierte Arrays anfordert, um die Daten in eine `list()` von Einzelvariablen einzulesen:

```
$sql = "SELECT ht_name, st_name FROM rb_hotels INNER JOIN rb_staedte
        ON ht_stadt=st_nr ORDER BY st_name, ht_name";
$query = $conn->query ($sql);
while (list ($hotel, $stadt) = $query->fetch (PDO::FETCH_NUM)) {
  echo ("$hotel, $stadt <br />\n");
}
```

Die etwas eigenwillige Schreibweise mit dem Modus PDO::FETCH_LAZY sieht dagegen so aus:

```
$sql = "SELECT ht_name, st_name FROM rb_hotels INNER JOIN rb_staedte
        ON ht_stadt=st_nr ORDER BY st_name, ht_name";
$query = $conn->query ($sql);
while ($row = $query->fetch (PDO::FETCH_LAZY)) {
  echo ($row->ht_name.", ".$row->st_name."<br />\n");
}
```

Übrigens können Sie den Modus auch vor dem ersten Aufruf von fetch() festlegen; dazu dient die Methode setFetchMode(). Das folgende Beispiel stellt den Modus PDO::FETCH_ASSOC ein:

```
$query = $conn->query ($sql);
$query->setFetchMode (PDO::FETCH_ASSOC);
```

Neben fetch() besitzt PDO noch die Methode fetchAll(), die alle Ergebnisdatensätze als Array ausliest. Die einzelnen Elemente dieses Arrays sind auch wieder Arrays, aus denen Sie gemäß dem eingestellten Fetch-Modus die einzelnen Felder lesen können. Hier ein Beispiel, das alle Flughäfen sowie ihre Kürzel und Städte auf diese Weise auflistet, zur Abwechslung im Modus PDO::FETCH_ASSOC:

```
try {
  $conn = new PDO ("mysql:host=localhost;dbname=reisebuero",
                   $user, $pass, array
                   (PDO::ATTR_PERSISTENT => 1,
                    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
  $sql = "SELECT ap_name, ap_kuerzel, st_name
          FROM rb_airports INNER JOIN rb_staedte
          ON ap_stadt=st_nr
          ORDER BY st_name ASC";
  // Abfrage senden
  $query = $conn->query($sql);
  // Fetch-Modus auf assoziative Arrays setzen
  $query->setFetchMode (PDO::FETCH_ASSOC);
  // alle Ergebnisdatensätze holen
  $rows = $query->fetchAll();
  // das Ergebnis-Array in einer Schleife auslesen
  foreach ($rows as $row) {
    echo ($row['ap_name']." (".$row['ap_kuerzel']."),
          ".$row['st_name']."<br />\n");
  }
  $conn = null;
} catch (PDOException $e) {
  echo ("Fehler: ".$e->getMessage());
}
```

Wenn Sie Auswahlabfragen in Schleifen ineinander verschachteln möchten, müssen Sie die äußere auf jeden Fall mittels fetchAll() auslesen; PDO ist nicht in der Lage, mehrere Abfragen im Speicher zu puffern. Das komplette Listing in Beispiel 8-3 illustriert dies, indem es die Städte und ihre Länder als Überschriften und die jeweils in ihnen befindlichen Hotels als Aufzählungen ausgibt:

Beispiel 8-3: Liste der Städte und ihrer Hotels mit verschachtelten PDO-Abfragen

```
<?php

  // Funktion für PDO-Verbindungsaufbau
  function pdo_conn () {
    // Verbindungsparameter
    $host = "localhost";
    $user = "rbuser";
    $pass = "R3153n";
    $db   = "reisebuero";
    try {
      $conn = new PDO ("mysql:host=$host; dbname=$db", $user, $pass,
              array (PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                     PDO::ATTR_PERSISTENT => 1));
    } catch (PDOException $e) {
      echo ("FEHLER: ".$e->getMessage());
      return null;
    }
    return $conn;
  }

?>
<html>
<head>
<title>St&auml;dte und Hotels, PDO</title>
</head>
<body>
<?php

  try {
    // Erste (äußere) Verbindung herstellen
    $conn1 = pdo_conn();
    // Städte und Länder auslesen und anzeigen
    $sql1 = "SELECT st_nr, st_name, la_name
             FROM rb_staedte INNER JOIN rb_laender
             ON st_land=la_nr ORDER BY la_name, st_name";
    $query1 = $conn1->query ($sql1);
    $query1->setFetchMode (PDO::FETCH_NUM);
    // Alle Datensätze als verschachteltes Array auslesen
    $rows = $query1->fetchAll();
    foreach ($rows as $row) {
      list ($stadt_nr, $stadt, $land) = $row;
      echo ("<h2>$stadt, $land</h2>\n");
      echo ("<ul>\n");
      // Zweite (innere) Verbindung herstellen
      $conn2 = pdo_conn();
      // Hotels der aktuellen Stadt auslesen und anzeigen
      $sql2 = "SELECT ht_name FROM rb_hotels
               WHERE ht_stadt=$stadt_nr";
      $query2 = $conn2->query ($sql2);
      $query2->setFetchMode (PDO::FETCH_LAZY);
      while ($row = $query2->fetch()) {
```

Beispiel 8-3: Liste der Städte und ihrer Hotels mit verschachtelten PDO-Abfragen (Fortsetzung)

```
        echo ("<li>".$row->ht_name."</li>\n");
      }
      // Innere Verbindung schließen
      $conn2 = null;
      echo ("</ul>\n");
    }
    // Äußere Verbindung schließen
    $conn1 = null;
  } catch (PDOException $e) {
    echo ("Fehler: ".$e->getMessage());
  }

?>
</body>
</html>
```

Zu guter Letzt noch ein Wort zu Änderungsabfragen in PDO. Wenn Sie diese mittels `query()` senden, können Sie anschließend die Methode `rowCount()` des `PDOStatement`-Objekts aufrufen, um die Anzahl der betroffenen Datensätze zu ermitteln. Alternativ können Sie Änderungsabfragen mithilfe der Methode `exec()` absetzen; das Ergebnis ist dann sofort die Anzahl. Hier beide Varianten im Vergleich (der Einzelzimmerpreis des Hotels Nummer 1 wird mithilfe von `query()` vorübergehend erhöht und dann mittels `exec()` wieder zurückgesetzt):

```
// Den aktuellen Preis zunächst auslesen
$query = $conn->query ("SELECT ht_ezpreis FROM rb_hotels
                        WHERE ht_nr=1");
list ($preis) = $query->fetch (PDO::FETCH_NUM);
echo ("Aktueller Preis: $preis <br />\n");
// Abfrageressource freigeben, um Verbindung wiederzuverwenden
$query = null;
// Änderungsabfrage mit query() senden
$query = $conn->query ("UPDATE rb_hotels
                        SET ht_ezpreis=ht_ezpreis + 10
                        WHERE ht_nr=1");
$affected = $query->rowCount();
echo ("$affected Zeile(n) ge&auml;ndert.<br />");
// Änderungsabfrage mit exec() senden
$affected = $conn->exec ("UPDATE rb_hotels
                          SET ht_ezpreis=$preis
                          WHERE ht_nr=1");
echo ("$affected Zeile(n) ge&auml;ndert.<br />");
```

Prepared Statements

Wie bereits in Kapitel 7 erwähnt, bietet PHP Data Objects eine eigene Schreibweise für Prepared Statements. Ihr Einsatz ist empfehlenswert, wenn eine Abfrage mehrmals eingesetzt werden soll. Um ein Prepared Statement zu erzeugen, wird die Methode `prepare()` verwendet. Das folgende Beispiel speichert eine Auswahlabfrage nach den Städten und ihren Ländern in einem Prepared Statement:

```
$query = $conn->prepare ("SELECT st_name, la_name
                          FROM rb_staedte INNER JOIN rb_laender
                          ON st_land=la_nr
                          ORDER BY la_name, st_name");
```

Ein Prepared Statement wird mithilfe seiner Methode `execute()` ausgeführt. Wenn es sich um eine Auswahlabfrage handelt, können die Ergebnisdatensätze anschließend wie gehabt mittels `fetch()` oder `fetchAll()` ausgelesen werden – für das obige Beispiel etwa so:

```
$query->execute();
while ($row = $query->fetch (PDO::FETCH_LAZY)) {
  echo ($row->st_name.", ".$row->la_name."<br />\n");
}
```

Noch interessanter werden Prepared Statements dadurch, dass Sie Parameter einfügen können, die sich bei jedem Aufruf von `execute()` ersetzen lassen. Diese Parameter können entweder als `?` oder als `:Parametername` geschrieben werden. Anschließend wird beim `execute()`-Aufruf ein Array mit ihren konkreten Werten übergeben, oder aber Sie binden mittels `bindParam()` eine Variable (und nicht etwa ihren Wert!) an einen der Parameter. Hier zunächst ein Beispiel mit einer Version der obigen Abfrage, in der ein einzelnes Land als Parameter angegeben werden kann:

```
$query = $conn->prepare ("SELECT st_name, la_name
                          FROM rb_staedte INNER JOIN rb_laender
                          ON st_land=la_nr WHERE la_nr=?
                          ORDER BY la_name, st_name");
// nur die Städte aus Deutschland auswählen
$query->execute (array (1));
// Ergebnisse auslesen wie bisher ...
```

Wenn Sie statt der Fragezeichen benannte Parameter wählen, wird für `execute()` ein assoziatives Array benutzt, in dem die Parameternamen die Schlüssel bilden:

```
$query = $conn->prepare ("SELECT st_name, la_name
                          FROM rb_staedte INNER JOIN rb_laender
                          ON st_land=la_nr WHERE la_nr=:land
                          ORDER BY la_name, st_name");
// Nur die Städte aus Deutschland auswählen
$query->execute (array (':land' => 1));
// Ergebnisse auslesen wie bisher ...
```

Die Methode `bindParam()` besitzt folgende Syntax:

```
$query->bindParam (Parameter, $Variable);
```

Fragezeichenparameter werden dabei ab 1 durchnummeriert, während benannte Parameter als Strings geschrieben werden. Hier der relevante Teil der weiter oben gezeigten Hotelpreisänderung mit ?:

```
$query = $conn->prepare ("UPDATE rb_hotels
                          SET ht_ezpreis=?
                          WHERE ht_nr=1");
$query->bindParam(1, $preis);
```

```
$preis += 10;
$query->execute();
$affected = $query->rowCount();
echo ("$affected Zeile(n) ge&auml;ndert.<br />");
$preis -= 10;
$query->execute();
$affected = $query->rowCount();
echo ("$affected Zeile(n) ge&auml;ndert.<br />");
```

Mit einem benannten Parameter sieht derselbe Codeteil so aus:

```
$query = $conn->prepare ("UPDATE rb_hotels
                          SET ht_ezpreis=:preis
                          WHERE ht_nr=1");
$query->bindParam(':preis', $preis);
$preis += 10;
$query->execute();
$affected = $query->rowCount();
echo ("$affected Zeile(n) ge&auml;ndert.<br />");
$preis -= 10;
$query->execute();
$affected = $query->rowCount();
echo ("$affected Zeile(n) ge&auml;ndert.<br />");
```

Clientseitiges Scripting mit JavaScript und Ajax

PHP ist eine *serverseitige Skriptsprache*. PHP-Skripten werden also bereits auf dem Webserver verarbeitet; das Ergebnis sind statische HTML-Dokumente (manchmal auch andere Formate wie Bilder oder PDF-Dateien), die an den Browser geliefert und von diesem angezeigt werden. Um mehr Dynamik in Ihre Webanwendungen zu bringen, können Sie zusätzlich *clientseitiges Scripting* verwenden.

Die von allen wichtigen Browsern (in leicht unterschiedlichen Dialekten) verstandene Sprache für diesen Einsatzzweck heißt *JavaScript*. Sie wurde 1995 von Netscape eingeführt und seitdem ständig weiterentwickelt. Über eine Objekthierarchie können damit wichtige Aspekte von Browserfenster und Dokument im laufenden Betrieb modifiziert werden. Seit 1999 unterstützen die gängigsten Browser auch das vom WWW-Konsortium (W3C) standardisierte *Document Object Model* (DOM), das den gesamten Inhalt einer Webseite als Baumstruktur betrachtet, in der Sie Inhalte beliebig einzufügen, entfernen oder modifizieren können.

Die aktuellste Neuentwicklung ist die *Ajax*-Technik. Durch die Kombination einiger Fähigkeiten, die einzeln schon seit einigen Jahren verfügbar sind, ermöglicht sie den Austausch einzelner Seitenbestandteile mithilfe separater HTTP-Anfragen. Ein serverseitiges Skript liefert dabei die neuen Inhalte, typischerweise aus einer Datenbank. Die Anfragen finden asynchron statt, so dass der Browser nicht passiv auf ihr Ergebnis wartet, sondern den Nutzer normal weiterarbeiten (zum Beispiel scrollen oder ein Formular ausfüllen) lässt.

JavaScript

Um JavaScript in einem HTML-Dokument unterzubringen, wird es in ein `<script>`-Tag verschachtelt, das wie folgt aussieht:

```
<script language="JavaScript" type="text/javascript">
...
</script>
```

Soll etwas gleich beim Laden des Dokuments an einer bestimmten Stelle ausgegeben werden, kann ein solches Konstrukt im Body stehen, ansonsten wird es eher in den Head gesetzt, um Funktionen zu definieren. Hier ein erstes kurzes Komplettbeispiel, das das Ergebnis einer Berechnung im Body anzeigt:

```
<html>
<head>
<title>Erstes JavaScript-Beispiel</title>
</head>
<body>
<h1>Berechnung durch JavaScript</h1>
<script language="JavaScript" type="text/javascript">

  ergebnis = 7 * 6;
  document.write ("7 * 6 = " + ergebnis);

</script>
</body>
</html>
```

Bereits dieses kurze Beispiel zeigt einige wichtige Eigenschaften von JavaScript:

1. Die Syntax ist derjenigen von PHP recht ähnlich.
2. Variablennamen beginnen nicht mit einem Dollarzeichen, sondern stets mit einem Buchstaben oder Unterstrich.
3. Aus diesem Grund ist innerhalb von Strings keine Variablensubstitution möglich. Zur Verkettung von Strings wird zudem kein Punkt wie in PHP verwendet, sondern ein Pluszeichen.
4. `document` ist ein Objekt, das das aktuelle HTML-Dokument repräsentiert. Seine Methode `write()` schreibt einfachen Text oder HTML an die aktuelle Position im Dokument – ähnlich wie die PHP-Funktion `echo()`, aber mit dem wichtigen Unterschied, dass diese bereits auf dem Server ihre Arbeit erledigt. Beachten Sie aber, dass `document.write()` nicht verwendet werden kann, um Inhalte nach dem erstmaligen Aufbau des Dokuments hinzuzufügen oder auszutauschen; dafür ist DOM zuständig.

Der üblichere Weg, JavaScript-Code aufzurufen, besteht in der Verwendung sogenannter *Event-Handler*. Dabei handelt es sich um spezielle Attribute von HTML-Tags. Beispielsweise können Sie dem `<body>`-Tag die Attribute `onload` und `onunload`

hinzufügen, um JavaScript unmittelbar nach dem Laden der aktuellen beziehungsweise als Letztes vor dem Laden einer anderen Seite auszuführen, beispielsweise:

```
<html>
<head>
<title>JavaScript-Gru&szlig; und -Abschied</title>
</head>
<body onload="alert ('Herzlich Willkommen!');"
 onunload="alert ('Auf wiedersehen!');">
<a href="http://www.mysql.com">Infos &uuml;ber MySQL</a>
</body>
</html>
```

Sobald diese Seite geladen wird, erscheint ein Pop-up-Dialog mit dem Gruß »Herzlich willkommen!«. Erst wenn Sie *OK* anklicken, können Sie die Seite verwenden. Auch beim Klick auf den Link *Infos über MySQL* oder bei Eingabe einer neuen URL wird ein solcher `alert()`-Dialog ausgeführt.

Die restlichen wichtigen Event-Handler werden in HTML-Formularen eingesetzt. Für Textfelder stehen zum Beispiel die Event-Handler `onchange` (Textänderung), `onkeyup` (Tastendruck), `onfocus` (Aktivierung per Mausklick oder Tab) und `onblur` (Deaktivierung) zur Verfügung.

Auch ein spezielles Element mit dem Schema

```
<input type="button" value="Beschriftung" />
```

kommt im Zusammenhang mit JavaScript häufig zum Einsatz. Es sieht aus wie ein Submit- oder Reset-Button. Es besitzt den Handler `onclick`, der beim Anklicken dieser Schaltfläche aktiviert wird.

Innerhalb der Anführungszeichen des Event-Handler-Werts gilt die JavaScript-Syntax. Wenn Sie darin wiederum Anführungszeichen benötigen, wie in den obigen `alert()`-Beispielen, müssen Sie einfache verwenden. In der Regel schreibt aber niemand längere Abfolgen von JavaScript-Anweisungen direkt in den Event-Handler, sondern ruft eine selbst geschriebene Funktion auf. Diese wird mithilfe des Schlüsselworts `function` definiert und besitzt dieselbe Syntax wie die weiter oben beschriebenen PHP-Funktionen.

Das folgende Beispiel stellt ein Formular bereit, mit dem Sie eine Suchanfrage an Google senden können (das Google-Suchskript erwartet einen Parameter namens `q` mit dem Suchbegriff, so dass das Textfeld hier diesen Namen erhält). Vor dem Versenden stellt die JavaScript-Funktion `testForm()` allerdings erst fest, ob überhaupt etwas eingegeben wurde, und beschwert sich andernfalls. Hier zunächst der komplette Code:

```
<html>
<head>
<title>Google-Suchanfrage senden</title>
<script language="JavaScript" type="text/javascript">
```

```
  // Funktion testForm(): Formular vor dem Versand überprüfen
  function testForm() {
    // Ist das Textfeld leer?
    if (document.formular.q.value == "") {
      // Warnung ausgeben
      alert ("Bitte zuerst Ihren Suchbegriff eingeben!");
      // Eingabefokus auf das Textfeld setzen
      document.formular.q.focus();
    } else {
      // Es wurde etwas eingegeben; Formular abschicken
      document.formular.submit();
    }
  }

</script>
</head>
<body>
<form name="formular" action="http://www.google.de/search"
 method="GET">
<input type="text" name="q" />
<input type="button" value="Google-Suche" onclick="testForm();" />
</form>
</body>
</html>
```

Wenn Sie dieses Skript ausführen und den Button *Google-Suche* anklicken, ohne etwas einzugeben, erscheint ein `alert()`. Anschließend wird der Cursor automatisch in das Textfeld gesetzt, was Sie aus Gründen der Benutzerfreundlichkeit bei jeder Formularüberprüfung veranlassen sollten. Damit das Formular nicht in jedem Fall versandt wird, ist die Schaltfläche kein Submit-Button, sondern ein einfacher Button. Sofern das Textfeld doch einen Inhalt besitzt, wird die Formular-Methode `submit()` aufgerufen, um das Formular per JavaScript zu versenden.

Das Skript zeigt zusätzlich die einfachste Möglichkeit, auf Formulare und ihre Elemente zuzugreifen: Wenn Sie ihnen jeweils ein `name`-Attribut zuweisen, können Sie sie in der Form `document.Formularname.Elementname` ansprechen. Auf diese Weise stehen Ihnen ihre Eigenschaften (zum Beispiel `value` für den Inhalt eines Textfelds) und Methoden wie etwa `Textfeld.focus()` oder `Formular.submit()` zur Verfügung.

Der Zugriff auf andere Bestandteile des Dokuments ist nicht ganz so einfach wie der auf Formulare, wie der nachfolgende Abschnitt über DOM zeigt.

Das Document Object Model

Etwa 1997 begannen die Browserhersteller damit, erweiterte JavaScript-Funktionalität unter dem Namen »Dynamic HTML« einzubauen. Damit konnte erstmals auch die Struktur einer bereits bestehenden Webseite nachträglich geändert werden. Das Ärgerliche war nur, dass es kein einheitliches Modell gab – Microsoft und Netscape gingen völlig verschiedene Wege.

Das W3C veröffentlichte daraufhin eine eigene Spezifikation, das *Document Object Model* (DOM). Inzwischen können Sie davon ausgehen, dass 99% aller Webuser einen Browser verwenden, der DOM versteht, so dass die klassischen Lösungen der Browserhersteller praktisch keine Rolle mehr spielen. DOM ist übrigens für verschiedene Programmiersprachen verfügbar und kann nicht nur HTML-, sondern auch beliebige XML-Dokumente verarbeiten.

DOM betrachtet das jeweilige Dokument als Baumstruktur, dessen Wurzel das äußerste Tag (daher auch Wurzelelement genannt) bildet. Bei einem HTML-Dokument ist dies das Tag `<html>`. Betrachten Sie exemplarisch die folgende Gegenüberstellung eines einfachen HTML-Dokuments und seiner Baumstruktur:

```
<html>                              +  Wurzel: Element "html"
                                    |
  <head>                            +--+  Element "head"
                                    |  |
    <title>                         |  +--+  Element "title"
                                    |     |
      DOM-Baum                      |     +--+  Text "DOM-Baum"
</title></head>                     |
<body>                              +--+  Element "body"
                                       |
    Text und Elemente sind             +--+  Text "Text und ..."
                                       |
    <i>                                +--+  Element "i"
                                          |
      Knoten                              +--+  Text "Knoten"
</i></body></html>
```

Jedes Element (HTML-Tag) und jedes Textelement wird als *Knoten* (englisch *Node*) bezeichnet. Die Knoten bilden eine hierarchische Struktur. Auf die innerhalb eines Knotens verschachtelten Unterknoten können Sie mithilfe des Arrays `childNodes` zugreifen. Der äußerste Knoten ist `document`; dieser hat einen Kindknoten, nämlich das Element `html`, und darunter sind alle anderen Tags und Textelemente als Knoten verschachtelt.

Das Tag `<i>` im obigen Beispielbaum wäre somit `document.childNodes[0].childNodes[1].childNodes[1]`. Für `childNodes[0]` gibt es alternativ die Schreibweise `firstChild`. Die Anzahl der Kinder eines Knotens ermitteln Sie mithilfe der Eigenschaft `Knoten.childNodes.length`, und die Methode `Knoten.hasChildNodes()` verrät, ob ein Knoten überhaupt Kindknoten besitzt (`true`) oder nicht (`false`).

Jeder Knoten besitzt die Eigenschaft `nodeType`, die den Knotentyp numerisch angibt. Die wichtigsten Typen sind 1 für ein HTML-Element, 3 für Text und 9 für das Dokument. Der Tag-Name eines Elementknotens kann mittels `nodeName` ausgelesen werden, der Inhalt eines Textknotens mithilfe von `nodeValue`.

Anstatt sich von `document` aus durch die Kind- und »Enkel«-Knoten zu hangeln, können Sie auch die folgenden beiden Methoden einsetzen, um direkt auf bestimmte Elemente zuzugreifen:

- `document.getElementById("Bezeichner")` wählt ein Element aus, dem Sie mithilfe des HTML-Attributs `id` eine eindeutige Bezeichnung zugewiesen haben. Wenn Sie beispielsweise ein Element als

  ```
  <div id="hauptmenue">...</div>
  ```

  kennzeichnen, erfolgt der Zugriff auf diesen Bereich mittels

  ```
  document.getElementById("hauptmenue")
  ```

- `document.getElementsByTagName("TagName")` liefert ein Array aller Elemente, die denselben Tag-Namen besitzen. Das folgende Konstrukt greift also auf den ersten Absatz (`<p>...</p>`) des Dokuments zu:

  ```
  document.getElementsByTagName("p")[0]
  ```

Die so ermittelten Elemente speichern Sie am einfachsten in Variablen. Anschließend können Sie nach Bedarf wieder auf ihre Kindknoten zugreifen.

Die folgende Beispieldatei enthält vier Absätze. Wenn Sie in das Textfeld eine Absatznummer eingeben und auf *Anzeigen* klicken, wird der Text des entsprechenden Absatzes (oder eine Fehlermeldung bei einem nicht existierenden Absatz) in einem separaten Bereich angezeigt:

```
<html>
<head>
<title>DOM-Beispiel: Absatz-Auswahl</title>
<style type="text/css">

  #anzeige {
    color: #FF0000;
    font-style: italic
  }

</style>
<script language="JavaScript" type="text/javascript">

  function absatzWahl() {
    // Nummer auslesen und Wertebereich anpassen
    nr = parseInt (document.f.zahl.value);
    if (nr < 0) {
      nr = 0;
    }
    // Array aller Absatz-Elemente speichern
    absaetze = document.getElementsByTagName("p");
    ausgabe = "";
    // Absatz vorhanden?
    if (absaetze.length >= nr) {
      // Text des gewählten Absatzes speichern
      ausgabe = absaetze[nr - 1].firstChild.nodeValue;
```

```
    } else {
      // Fehlermeldung speichern
      ausgabe = "*** Fehler: Absatz nicht vorhanden! ***";
    }
    // Das Element mit der ID "anzeige" speichern
    anzeigeblock = document.getElementById("anzeige");
    // Den bisherigen Text löschen
    anzeigeblock.removeChild (anzeigeblock.firstChild);
    // Einen Textknoten mit dem neuen Text erzeugen
    txt = document.createTextNode (ausgabe);
    // Im richtigen Bereich einhängen
    anzeigeblock.appendChild (txt);
  }

</script>
</head>
<body>
<p>
DOM ist ein Konzept zum Lesen und &Auml;ndern der Struktur und der Inhalte eines
HTML-Dokuments.</p>
<p>Das Verfahren wurde 1999 vom W3C standardisiert.</p>
<p>Heute bildet DOM eine der Grundlagen von Ajax.</p>
<p>Im n&auml;chsten Beispiel werden Inhalte eingef&uuml;gt, gel&ouml;scht und
modifiziert.</p>
<form name="f">
Absatz Nr.: <input type="text" name="zahl" />
<input type="button" value="Anzeigen" onclick="absatzWahl();" />
</form>
<div id="anzeige"> </div>
</body>
</html>
```

Das Skript demonstriert einige weitere wichtige DOM-Aspekte – insbesondere die dynamische Änderung des Dokumentinhalts. Wie Sie sehen, wird zunächst `Knoten.removeChild()` verwendet, um den bisherigen Text des `<div>`-Elements `anzeige` zu löschen. Die Methode `document.createTextNode ("Text")` erzeugt daraufhin einen neuen Textknoten, der mittels `Knoten.appendChild()` in das `<div>` eingefügt wird. Das Beispiel zeigt auch, dass Sie die eindeutige ID eines Elements zur CSS-Formatierung (hier rot und kursiv) verwenden können.

Ajax

Der Begriff *Ajax* wird manchmal als Abkürzung für *Asynchronous JavaScript And XML* betrachtet; der Erfinder des Worts, Jesse James Garrett, ist allerdings anderer Meinung. Für ihn ist Ajax nur ein Wort und kein Akronym. Jedenfalls sind asynchrone Anfragen, JavaScript und XML wichtige Komponenten dieser Technologie.

Wie Sie wissen – und bisher in diesem Buch gesehen haben –, lädt eine herkömmliche Webanwendung bei jedem Arbeitsschritt eine vollständige HTML-Seite vom Server nach. JavaScript und DOM können zwar für clientseitige Dynamik sorgen, ermöglichen aber selbst zunächst kein Nachladen von Serverdaten.

Hier kommt das `XMLHttpRequest`-Objekt, die Grundlage von Ajax, ins Spiel: Ein solches Objekt kann eine HTTP-Anfrage an einen Webserver senden und braucht dabei noch nicht einmal untätig auf dessen Antwort zu warten. Stattdessen wird nach dem Eintreffen der Antwort automatisch eine Funktion aufgerufen, deshalb wird eine solche Anfrage als asynchron bezeichnet. Die Serverantwort ist manchmal einfacher Text, häufig aber XML, das mithilfe von DOM-Operationen zerlegt werden kann. Auch zum Einfügen der Antwortdaten in das bestehende Dokument kommt in der Regel DOM zum Einsatz.

Der erste Schritt beim Schreiben einer Ajax-Anwendung ist die Erzeugung des besagten Anfrageobjekts. Da es in verschiedenen Browsern unterschiedlich heißt, können Sie zum Beispiel folgenden Code dafür verwenden:

```
// Referenzvariable deklarieren
var anfrage = null;
try {
  // Firefox, Opera, IE7
  anfrage = new XMLHttpRequest();
} catch (err_new) {
  try {
    // IE 6.x
    anfrage = new ActiveXObject("Msxml2.XMLHTTP");
  } catch (err_ie6) {
    try {
      // IE 5.x
      anfrage = new ActiveXObject("Microsoft.XMLHTTP");
    } catch (err_noajax) {
      // Ajax-unfähiger Browser
      anfrage = null;
    }
  }
}
if (anfrage == null) {
  // Fehlermeldung
  alert ("Ihr Browser versteht leider kein Ajax.");
}
```

Die Variable `anfrage` enthält danach ein Ajax-Anfrageobjekt, sofern der verwendete Browser eines unterstützt. Als Nächstes wird die Ajax-Anfrage an ein serverseitiges Skript vorbereitet und abgesendet. Die notwendigen Schritte sehen formal wie folgt aus:

```
anfrage.open (methode, url, asynchron);
anfrage.onreadystatechange = callbackFunktion;
anfrage.send (Body);
```

Die Methode open benötigt drei Argumente:

- methode ist die bereits bekannte HTTP-Methode, also "GET" oder "POST"
- url ist die URL des angesprochenen serverseitigen Skripts; bei GET-Anfragen werden oft Daten in der Form ?Feld1=Wert1&Feld2=Wert2... angehängt.
- asynchron ist true für eine asynchrone oder false für eine synchrone HTTP-Anfrage; Letztere wartet untätig auf die Serverantwort, was manchmal nützlich ist, wenn Anfangsdaten benötigt werden, ohne die eine Anwendung nicht funktioniert.

anfrage.onreadystatechange gibt den Namen der Funktion an, die beim Eintreffen der Serverantwort aufgerufen wird. Zum Schluss wird send() aufgerufen, um die Anfrage abzuschicken. Bei POST-Anfragen wird ein Body mitgeschickt, standardmäßig ebenfalls im Format application/x-www-urlencoded, das heißt Feld1=Wert1&Feld2=Wert2...; bei GET-Anfragen ignoriert der Server den Body, so dass Sie bei diesen null schreiben können.

Das folgende Beispiel sendet eine asynchrone GET-Anfrage mit einem Feld namens such an das Skript *staedte.php*; wenn die Antwort vorliegt, soll die Funktion stadtListe() aufgerufen werden:

```
anfrage.open ("GET", "staedte.php?such=" + such, true);
anfrage.onreadystatechange = stadtListe;
anfrage.send (null);
```

Die unter onreadystatechange angegebene Funktion wird bei jeder Änderung der Anfrageeigenschaft readyState aufgerufen. Dieser *Bereitschaftszustand* kann Werte von 0 bis 4 annehmen, wobei 4 dem Eintreffen der Serverantwort entspricht. Zusätzlich sollten Sie die Eigenschaft anfrage.status überprüfen; sie enthält den Statuscode der HTTP-Antwort. Akzeptabel ist nur 200 (OK); 404 (Not Found) weist dagegen zum Beispiel darauf hin, dass das angesprochene Skript nicht existiert. Deshalb hat praktisch jede Ajax-Callback-Funktion folgende Grundstruktur:

```
function stadtListe() {
  // Liegt die Antwort vor?
  if (anfrage.readyState == 4) {
    // Ist die Antwort in Ordnung?
    if (anfrage.status == 200) {
      // ... Antwort verarbeiten ...
    } else {
      alert ("Leider ist ein Fehler aufgetreten.");
    }
  }
}
```

Die Antwort wird aus der Eigenschaft anfrage.responseText gelesen, wenn es sich um einfachen Text handelt. XML-basierte Antworten stehen dagegen in anfrage.responseXML zur Verfügung; es handelt sich um den Wurzelknoten eines DOM-Baums, der sich mithilfe der weiter oben vorgestellten DOM-Konstrukte durchsuchen lässt.

Das folgende einfache Beispiel stellt ein Texteingabefeld zur Verfügung. Bei jeder Änderung der Eingabe wird eine Ajax-Anfrage an das PHP-Skript *staedte.php* gesendet, das eine Liste aller zur aktuellen Eingabe passenden Städte aus der Reisebüro-Datenbank zurückliefert. Das Prinzip ähnelt Google Suggest (*http://labs.google.com/suggest/*), wo die jeweils am häufigsten vorkommenden Suchbegriffe zur bisherigen Eingabe angezeigt werden.

In Beispiel 8-4 sehen Sie zunächst die HTML-Datei mit sämtlichem JavaScript. Auf der CD-ROM finden Sie diese Anwendung im Verzeichnis *beispiele/ajax-demo*.

Beispiel 8-4: Die Ajax-basierte Städteliste stadtliste.php

```
<html>
<head>
<title>Dynamische St&auml;dteliste</title>
<script language="JavaScript" type="text/javascript">

  // Ajax-Anfrage-Objekt erzeugen

  // Globale Referenzvariable deklarieren
  var anfrage = null;
  try {
    // Firefox, Opera, IE7
    anfrage = new XMLHttpRequest();
  } catch (err_new) {
    try {
      // IE 6.x
      anfrage = new ActiveXObject("Msxml2.XMLHTTP");
    } catch (err_ie6) {
      try {
        // IE 5.x
        anfrage = new ActiveXObject("Microsoft.XMLHTTP");
      } catch (err_noajax) {
        // Ajax-unfähiger Browser
        anfrage = null;
      }
    }
  }
  if (anfrage == null) {
    // Fehlermeldung
    alert ("Ihr Browser versteht leider kein Ajax.");
  }

  // Anfragefunktion für die aktuelle Städteliste
  function sucheStadt() {
    // Formularfeld auslesen
    such = document.f.stadt.value;
    // Anfrage vorbereiten
    anfrage.open ("GET", "staedte.php?such=" + such, true);
    anfrage.onreadystatechange = stadtListe;
    anfrage.send (null);
  }
```

Beispiel 8-4: Die Ajax-basierte Städteliste stadtliste.php (Fortsetzung)

```
  // Städte auslesen und anzeigen
  function stadtListe() {
    // Liegt die Antwort vor?
    if (anfrage.readyState == 4) {
      // Ist die Antwort in Ordnung?
      if (anfrage.status == 200) {
        // DOM-Baum entgegennehmen
        antwort = anfrage.responseXML;
        // Alle <stadt>-Elemente auslesen
        // und in einem Array speichern
        staedtenamen = new Array();
        staedte = antwort.getElementsByTagName("stadt");
        for (i = 0; i < staedte.length; i++) {
          name = staedte[i].firstChild.nodeValue;
          staedtenamen.push (name);
        }
        // Bisherigen Inhalt der Liste leeren
        liste = document.getElementById("stadtliste");
        while (liste.hasChildNodes()) {
          liste.removeChild (liste.firstChild);
        }
        // Textknoten und Zeilenumbrüche für alle Städte
        // erzeugen und anhängen
        for (i in staedtenamen) {
          txt = document.createTextNode (staedtenamen[i]);
          br = document.createElement ("br");
          liste.appendChild (txt);
          liste.appendChild (br);
        }
      } else {
        alert ("Leider ist ein Fehler aufgetreten.");
      }
    }
  }

</script>
</head>
<body>
<form name="f">
Stadt:
<input type="text" name="stadt" onkeyup="sucheStadt();" />
</form>
<div id="stadtliste">St&auml;dte ...</div>
</body>
</html>
```

Die PHP-Datei *staedte.php*, die alle zur aktuellen Suchanfrage passenden Städte heraussucht und als XML-Dokument zurückgibt, wird in Beispiel 8-5 gezeigt. Sie verwendet die *mysqli*-Schnittstelle und wird hier unkommentiert abgedruckt, da Sie das verwendete PHP bereits kennen. Beachten Sie nur die `header()`-Anweisung, die

den `Content-type` vor der Auslieferung des eigentlichen Dokuments auf `text/xml` setzt. Mindestens im Internet Explorer funktioniert die Anwendung andernfalls nicht.

Beispiel 8-5: Die PHP-Datei staedte.php, die die jeweils aktuelle Städteliste im XML-Format liefert

```
<?php

  function cgi_param ($feld, $default="") {
    $var = $default;
    if (isset ($_GET[$feld]) && $_GET[$feld] != "") {
      $var = $_GET[$feld];
    } elseif (isset ($_POST[$feld]) && $_POST[$feld] != "") {
      $var = $_POST[$feld];
    }
    return $var;
  }

  $host = "localhost";
  $user = "rbuser";
  $pass = "R3153n";
  $db = "reisebuero";

  $conn = new mysqli ($host, $user, $pass, $db);

  $such = cgi_param ("such", "");

  $ausgabe = "<?xml version=\"1.0\" encoding=\"iso-8859-1\" ?>\n";
  $ausgabe .= "<staedte>\n";


  $query = $conn->query ("SELECT st_name FROM rb_staedte
                          WHERE st_name LIKE \"%$such%\"");
  while (list ($stadt) = $query->fetch_row()) {
    $ausgabe .= "<stadt>$stadt</stadt>\n";
  }

  $query->close();

  $ausgabe .= "</staedte>\n";

  header ("Content-type: text/xml; charset=iso-8859-1");
  echo ($ausgabe);

?>
```

Abbildung 8-1 zeigt die Anwendung im Einsatz, nachdem der Buchstabe *a* eingegeben wurde.

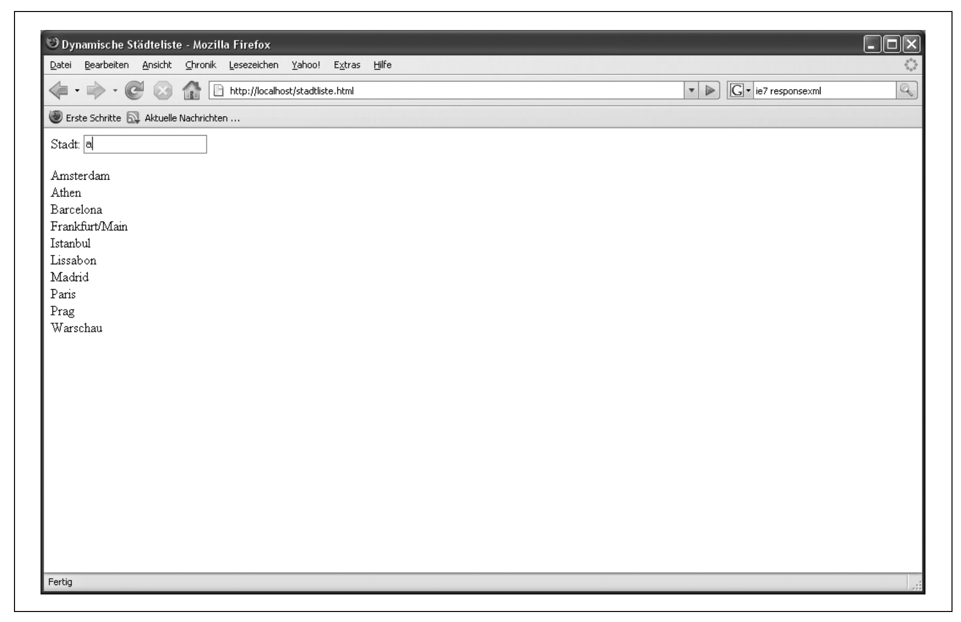


Abbildung 8-1: Die Ajax-Demoanwendung im Einsatz

Die Reisebüro-Anwendung

In diesem Abschnitt werden einige längere PHP-Skripten vorgestellt: ein Gästebuch, ein Diskussionsforum sowie einige wichtige Teile der Reisebuchungsanwendung. Die abgedruckten Beispiele verwenden ausschließlich die Schnittstelle *mysqli*, da diese für PHP 5 und MySQL 5 empfehlenswerter ist als das klassische *mysql*. Auf der beiliegenden CD-ROM sind diese und alle weiteren Beispiele aber in allen drei Varianten enthalten.

Jede Version der Anwendung besteht aus folgenden Dateien:

- *util.inc.php* – allgemein genutzte PHP-Funktionen
- *utils.js* – eine allgemein genutzte JavaScript-Funktion
- *index.php* – Startseite
- *login.php* – Benutzer-Login oder -Neuanmeldung
- *testlog.php* – Überprüfung des Logins
- *newuser.php* – Eintragung eines neuen Users in die Datenbank
- *logout.php* – Benutzerabmeldung
- *auskunft.php* – Suche nach Reiseangeboten
- *ergebnis.php* – Suchergebnisse mit Buchungsformular
- *buchung.php* – Eintrag der Buchung in die Datenbank

- *buchabschl.php* – Bestätigung oder Storno der Buchung
- *buchungen.php* – Übersicht über alle Buchungen des aktuellen Users
- *gast.php* – das Gästebuch
- *forum.php* – das Diskussionsforum
- *info.php* – Informationen über die Reiseziele und ihre Sehenswürdigkeiten
- *admin/index.php* – Liste der Buchungen aller User (Login *admin*, Passwort *reise*)
- *main.css* – allgemeines Stylesheet
- *logo.gif* – das Logo des Reisebüros

Bevor die Beispiele selbst behandelt werden, sollen drei wichtige Funktionen verallgemeinert und ausgelagert werden: das Auslesen von Formular- und Sessiondaten sowie das Herstellen einer Datenbankverbindung. Mithilfe der PHP-Funktion `include()` können Sie nämlich zusätzliche PHP-Dateien einbetten. Diese werden üblicherweise durch die doppelte Dateiendung *.inc.php* gekennzeichnet. Auch innerhalb solcher Include-Dateien müssen PHP-Anweisungen in `<?php ... ?>`-Blöcken stehen.

Die vollständige Datei *util.inc.php*, auf die sich alle hier abgedruckten Skripten beziehen, hat in der *mysqli*-Version folgenden Inhalt:

```
<?php

   function cgi_param ($feld, $default="") {
     $var = $default;
     if (isset ($_GET[$feld]) && $_GET[$feld] != "") {
       $var = $_GET[$feld];
     } elseif (isset ($_POST[$feld]) && $_POST[$feld] != "") {
       $var = $_POST[$feld];
     }
     return $var;
   }

   function session_param ($feld, $default="") {
      $var = $default;
      if (isset ($_SESSION[$feld]) && $_SESSION[$feld] != "") {
         $var = $_SESSION[$feld];
      }
      return $var;
   }

   function db_connect ($database, $user, $pass) {
     $host = "127.0.0.1";
     $conn = new mysqli ($host, $user, $pass, $database);
     return $conn;
   }

?>
```

Der nützliche Formulardatenleser `cgi_param()` wurde bereits in Kapitel 3 vorgestellt. `session_param()` erledigt die gleiche Aufgabe für die weiter oben vorgestellten Session-Variablen. Die Funktion `db_connect()` ist dagegen neu. Sie nimmt den Namen der gewünschten Datenbank, einen Benutzernamen und ein Passwort entgegen und liefert ein *mysqli*-Verbindungsobjekt zurück. Der Hostname ist dagegen auf *localhost* festgelegt. Wenn Sie möchten, können Sie natürlich auch Versionen schreiben, in denen mehr oder weniger dieser Parameter feststehen. Ein Aufruf der vorliegenden Version sieht für die Datenbank *reisebuero* und den im vorigen Abschnitt erstellten Benutzer *rbuser* so aus:

```
// Include-Datei einbinden
include ("util.inc.php");
// Verbindung zur Datenbank reisebuero herstellen
$conn = db_connect ("reisebuero", "rbuser", "R3153n");
```

Die PDO-Version der Funktion wurde im Wesentlichen bereits weiter oben gezeigt, wenn auch ohne variable Parameter. Deshalb sehen Sie hier die passende Fassung für die Beispielanwendung:

```
function db_connect ($database, $user, $pass) {
  $host = "127.0.0.1";
  try {
    $conn = new PDO ("mysql:host=$host;dbname=$db",
                     $user, $pass, array (
                     PDO::ATTR_PERSISTENT => 1,
                     PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION));
  } catch (PDOException $e) {
    echo ("Leider ist ein Datenbank-Verbindungsfehler
           aufgetreten.<br />");
    echo ("Fehlermeldung: ".$e->getMessage()."<br />");
    // keine Verbindung zurückgeben
    return null;
  }
  // Verbindung zurückgeben
  return $conn;
}
```

Die prozedurale *mysql*-Variante gibt statt des Objekts die numerische Verbindungs-ID zurück:

```
function db_connect ($database, $user, $pass) {
  $host = "127.0.0.1";
  $connID = mysql_connect ($host, $user, $pass);
  mysql_select_db ($database);
  return $connID;
}
```

Ein entsprechender Aufruf hat folgende Syntax:

```
include ("util.inc.php");
$connID = old_db_connect ("reisebuero", "rbuser", "R3153n");
```

Auch eine externe JavaScript-Datei namens *utils.js* wird in jede Datei eingebettet. Dies geschieht mithilfe des folgenden `<script>`-Tags:

```
<script language="JavaScript" type="text/javascript" src="utils.js">
</script>
```

Die Datei enthält nur eine Funktion namens `logout()`:

```
function logout() {
  var lo = confirm ("Wirklich ausloggen?");
  if (lo) {
    location.href="logout.php";
  }
}
```

Diese Funktion wird beim Klick auf den Hyperlink *Logout* aufgerufen, der nur angezeigt wird, wenn gerade ein Benutzer angemeldet ist. Die JavaScript-Funktion `confirm()` enthält im Unterschied zu `alert()` zwei Schaltflächen, die mit *OK* und *Abbrechen* beschriftet sind und beim Klick auf diese `true` beziehungsweise `false` zurückgeben. Mithilfe von `location.href` können Sie eine JavaScript-Weiterleitung zur angegebenen URL erzeugen. Der eigentliche Link ruft nur diese Funktion auf, was wie folgt funktioniert:

```
<a href="javascript:logout();">Logout</a>
```

Auf diese Weise lassen sich Sicherheitsabfragen realisieren, die beispielsweise auch beim Löschen von Datensätzen in PHP-Skripten sehr nützlich sind.

Das Gästebuch

Gästebücher sind auf Websites weit verbreitet; sie gestalten den Kontakt zwischen Unternehmen (oder privatem Website-Betreiber) und Site-Besuchern persönlicher. Auch die Reisebüro-Website soll mit einem Gästebuch ausgestattet werden.

Natürlich wäre es unpraktisch, die Gästebuchdaten in der großen »Geschäftsdatenbank« *reisebuero* aufzubewahren. Stattdessen wird zu diesem Zweck eine separate Datenbank angelegt. Starten Sie dazu den Kommandozeilenclient oder auch phpMyAdmin und erstellen Sie folgende Datenbank:

```
mysql> CREATE DATABASE gaestebuch;
```

Wenn Sie `mysql` verwenden, müssen Sie die Datenbank nun als Standard auswählen:

```
mysql> USE gaestebuch
```

Auch auf diese Datenbank soll der Benutzer *rbuser* Vollzugriff erhalten:

```
mysql> GRANT ALL PRIVILEGES ON gaestebuch.* TO rbuser@localhost;
mysql> FLUSH PRIVILEGES;
```

Zu guter Letzt wird die (einzige) Tabelle der neuen Datenbank erstellt. Sie enthält folgende Spalten: eine laufende Eintragsnummer als Primärschlüssel, Datum und

Uhrzeit des Eintrags, den Nickname des jeweiligen Eintragenden, dessen E-Mail-Adresse, einen Titel für den Eintrag sowie den eigentlichen Text.

```
mysql> CREATE TABLE gb_eintraege (
    ->    e_id INT AUTO_INCREMENT PRIMARY KEY,
    ->    e_datum DATETIME,
    ->    e_nick VARCHAR(40),
    ->    e_mail VARCHAR(40),
    ->    e_titel VARCHAR(100),
    ->    e_inhalt TEXT,
    ->    INDEX (e_titel)
    -> );
```

Nun geht es an die Entwicklung des Gästebuch-Skripts. Gästebücher bestehen aus mindestens drei verschiedenen »Betriebsmodi«: Einträge lesen, einen neuen Eintrag verfassen und den neuen Eintrag speichern. Diese lassen sich entweder als getrennte PHP-Dokumente realisieren, oder es wird – wie in der vorliegenden Lösung – ein Kriterium benötigt, das per `if`-Fallentscheidung zwischen diesen Aspekten unterscheidet und jeweils die erforderlichen Aktionen durchführt und Inhalte anzeigt.

In diesem Skript fungiert ein Parameterfeld namens `mode` als Unterscheidungskriterium. Zu Beginn des Skripts wird es ausgelesen. Je nachdem, welchen Wert es besitzt, werden die vorhandenen Einträge angezeigt (Wert `"r"` für *read*), das Formular für einen Neueintrag wird zur Verfügung gestellt (`"e"` für *entry*), der neue Eintrag wird gespeichert (`"s"` für *save*), oder eine Fehlermeldung wird ausgegeben (unbekannter Wert). Falls `mode` gar nicht vorhanden ist, wird praktischerweise das Lesen als Standardwert gesetzt. Schematisch besitzt das Skript also folgenden Aufbau:

```
$mode = cgi_param ("mode", "r");
if ($mode == "r") {
   // Einträge anzeigen
} elseif ($mode == "e") {
   // Formular für neuen Eintrag anzeigen
} elseif ($mode == "s") {
   // Formulardaten lesen und in Datenbank speichern
} else {
   // Fehlermeldung: unbekannter Modus
}
```

Der erste Abschnitt zeigt alle bisherigen Gästebucheinträge an. Dazu wird folgende SQL-Abfrage verwendet, die das Datum »deutsch« formatiert und die Einträge absteigend nach Datum, das heißt den neuesten zuerst, sortiert:

```
SELECT DATE_FORMAT(e_datum, '%d.%m.%Y, %H:%i'),
e_nick, e_mail, e_titel, e_inhalt
FROM gb_eintraege ORDER BY e_datum DESC
```

Sobald diese Abfrage wie üblich mittels `mysql->query()` abgesetzt ist, wird die Eigenschaft `$query->num_rows` ausgelesen, die die Anzahl der gefundenen Zeilen enthält. Falls sie den Wert 0 hat, wird die Meldung »Noch keine Einträge vorhanden« ausgegeben. Andernfalls werden die Datensätze per `fetch_row()`-Schleife ausgelesen und formatiert angezeigt.

Im Modus "e" (Neueintrag) wird der PHP-Block per ?> geschlossen, da das Formular statisches HTML ist. Die Versand-URL des Formulars ist wieder das Skript *gast.php* selbst. Der neue Modus "s" (Speichern) wird als `hidden`-Feld mit dem Formular versandt:

```
<form action="gast.php" method="post">
<input type="hidden" name="mode" value="s" />
...
</form>
```

Zum Speichern werden die Formulareinträge zunächst mithilfe der Funktion `cgi_param()` ausgelesen. Anschließend dient die folgende Abfrage dazu, sie in die Datenbank zu schreiben:

```
INSERT INTO gb_eintraege
(e_datum, e_nick, e_mail, e_titel, e_inhalt)
VALUES (NOW(), \"$nick\", \"$mail\", \"$titel\", \"$eintr\")
```

Die Escape-Sequenzen – durch Backslash geschützte Anführungszeichen – sind hier nur nötig, weil die Abfrage als String selbst in Anführungszeichen steht. Sie sind kein Bestandteil von SQL. Die MySQL-Funktion `NOW()` haben Sie bereits kennengelernt – hier dient sie dazu, den aktuellen Zeitpunkt in das Feld `e_datum` einzutragen.

Beispiel 8-6 zeigt das gesamte mit einigen Kommentaren versehene PHP-Skript *gast.php*.

Beispiel 8-6: gast.php – das Gästebuch des Reisebüros

```
<?php

   // Include-Datei laden
   include ("util.inc.php");

   // Datenbankverbindung herstellen
   $conn = db_connect ("gaestebuch", "rbuser", "R3153n");

   // Bereits eingeloggt?
   session_start();
   $user_nummer = session_param ("user", 0);

   // Aktuellen Modus ermitteln:
   // Lesen (r), Eintragen (e) oder Speichern (s)?
   // Standard ist Lesen
   $mode = cgi_param ("mode", "r");


?>

<html>
  <head>
    <title>Reiseb&uuml;ro: G&auml;stebuch</title>
```

Beispiel 8-6: gast.php – das Gästebuch des Reisebüros (Fortsetzung)

```
    <link rel="stylesheet" type="text/css" href="main.css" />
    <script language="JavaScript" type="text/javascript" src="utils.js"></script>
  </head>
  <body>
    <table border="0" width="750" align="center">
    <tr>
    <td>
 <br /><div align="center"><img src="logo.
gif" width="480" height="80" alt="EuroCityTravel" /></div><br />
    <!-- Einfache Navigationsleiste -->
    [
    <a href="index.php">Home</a>
    |
    <a href="auskunft.php">Reisesuche</a>
    |
    <a href="info.php">Touristeninfo</a>
    |
<?php

    if ($user_nummer > 0) {
      echo ("<a href=\"javascript:logout();\">Logout</a>");
    } else {
      echo ("<a href=\"login.php\">Login</a>");
    }

?>
    |
    <?php
      if ($user_nummer) {
        echo ("<a href=\"buchungen.php\">Buchungen</a> | ");
      }
    ?>
    G&auml;stebuch
    |
    <a href="forum.php">Forum</a>
    ]
    <!-- Ende der Navigationsleiste -->
<?php

   if ($mode == "r") {
      // Lesemodus: Einträge anzeigen

      echo ("<h2>G&auml;stebuch lesen</h2>");

      // Link für Neueintrag
      echo ("<a href=\"gast.php?mode=e\">Ins G&auml;stebuch eintragen</a><br /><br />");

      // Abfrage für alle Einträge - neuester zuerst
      $querytext = "SELECT DATE_FORMAT(e_datum, '%d.%m.%Y, %H:%i'),
                    e_nick, e_mail, e_titel, e_inhalt
                    FROM gb_eintraege
                    ORDER BY e_datum DESC";
```

Beispiel 8-6: gast.php – das Gästebuch des Reisebüros (Fortsetzung)

```
      $query = $conn->query ($querytext);

      // Anzahl der Einträge ermitteln
      $anzahl = $query->num_rows;

      if ($anzahl == 0) {
         // Meldung, falls keine Einträge vorhanden sind
         echo ("Noch keine Eintr&auml;ge vorhanden.<br />");
      } else {
         // alle Einträge anzeigen
         while (list ($datum, $nick, $mail, $titel, $text) =
                $query->fetch_row()) {
            // Nickname leer? -> "Anonymous"
            if ($nick == "") {
               $nick = "Anonymous";
            }
            // Titel leer? -> "[Ohne Titel]";

            if ($titel == "") {
               $titel = "[Ohne Titel]";
            }

            // Titel ausgeben
            echo ("<h3>$titel</h3>");

            // Nickname ausgeben
            if ($mail != "") {
               // E-Mail-Adresse vorhanden? -> Nickname als Link
               echo ("von <a href=\"mailto:$mail\">$nick</a>");
            } else {
               // nur Nickname
               echo ("von $nick");
            }

            // Datum ausgeben
            echo (", $datum<br /><br />");

            // Text und Trennlinie ausgeben
            echo ("$text<hr />");
         }
      }
   } elseif ($mode == "e") {
      // Schreibmodus: Eintragsformular anzeigen

?>
<h2>Neuer G&auml;stebucheintrag</h2>
<a href="gast.php?mode=r">Eintr&auml;ge lesen</a>
<form action="gast.php" method="post">
<input type="hidden" name="mode" value="s" />
<table border="0" cellpadding="4">
```

Beispiel 8-6: gast.php – das Gästebuch des Reisebüros (Fortsetzung)

```
  <tr>
    <td>Nickname:</td>
    <td><input type="text" name="nick" size="50" /></td>
  </tr>
  <tr>
    <td>E-Mail (optional):</td>
    <td><input type="text" name="mail" size="50" /></td>
  </tr>
  <tr>
    <td>Titel:</td>
    <td><input type="text" name="titel" size="50" /></td>
  </tr>
  <tr>
    <td valign="top">Ihr Eintrag:</td>
    <td><textarea name="eintr" cols="40" rows="7" wrap="virtual"></textarea></td>
  </tr>
  <tr>
    <td> </td>
    <td><input type="submit" value="Eintragen" />
  </tr>
</table>
<?php

   } elseif ($mode == "s") {
      // Speichermodus: neuen Eintrag speichern

      // Formularfelder auslesen
      $nick = cgi_param ("nick", "");
      $mail = cgi_param ("mail", "");
      $titel = cgi_param ("titel", "");
      $eintr = cgi_param ("eintr", "[kein Text]");

      // SQL-String für die Einfügeabfrage
      $querytext = "INSERT INTO gb_eintraege
                    (e_datum, e_nick, e_mail, e_titel, e_inhalt)
                    VALUES (NOW(), \"$nick\", \"$mail\", \"$titel\", \"$eintr\")";

      // Abfrage senden
      $conn->query ($querytext);

      // Hat es geklappt?
      if ($conn->affected_rows == 1) {
         echo ("<br /><br />Ihr Eintrag wurde erfolgreich hinzugef&uuml;gt.<br />
         <br />");
      } else {
         echo ("<br /><br />Aufgrund eines Fehlers konnte Ihr Eintrag leider nicht
         hinzugef&uuml;gt werden.<br /><br />");
      }

      echo ("<a href=\"gast.php?mode=r\">Eintr&auml;ge lesen</a>");
   } else {
```

Beispiel 8-6: gast.php – das Gästebuch des Reisebüros (Fortsetzung)

```
?>
<h2>Fehler</h2>
Ung&uuml;ltiger Zugriff auf das G&auml;stebuch.<br />
<a href="gast.php?mode=r">Zur&uuml;ck</a>
<?php

   }

?>
</td>
</tr>
</table>
</body>
</html>
```

In Abbildung 8-2 wird anhand einiger Gästebucheinträge der Modus "r" illustriert.



Abbildung 8-2: Das Gästebuch – Einträge lesen

Das Diskussionsforum

Noch beliebter als Gästebücher sind inzwischen Diskussionsforen, die den direkten Austausch zwischen verschiedenen Besuchern einer Website ermöglichen – interessante Foren sorgen dafür, dass Besucher immer gern wiederkommen.

Das Skript ist dem Gästebuch recht ähnlich. Auch hier werden die Modi verwendet, um zwischen den verschiedenen Aufgaben zu unterscheiden. Das Forum besitzt einen zusätzlichen Modus, nämlich eine nach Threads (aufeinander bezogenen Beiträgen) sortierte Liste aller Beiträge. Um zu bestimmen, zu welchem Thread ein Beitrag gehört, wird neben dem Primärschlüssel (der normalen ID) eine Parent-ID gespeichert – es handelt sich dabei einfach um die ID des beantworteten Beitrags. Neue Beiträge, die keine Antworten sind, erhalten die Parent-ID 0.

Bis auf das neue Feld *Parent-ID* ist die Struktur der Datenbanktabelle mit dem Gästebuch identisch. Erstellen Sie mithilfe der folgenden Anweisungen die neue Datenbank *forum*, die Tabelle *fo_eintraege* und eine vollständige Zugriffsberechtigung für den Benutzer *rbuser*:

```
mysql> CREATE DATABASE forum;
mysql> USE forum;
mysql> CREATE TABLE fo_eintraege (
    ->   f_id INT AUTO_INCREMENT PRIMARY KEY,
    ->   f_pid INT,
    ->   f_datum DATETIME,
    ->   f_nick VARCHAR(40),
    ->   f_mail VARCHAR(40),
    ->   f_titel VARCHAR(100),
    ->   f_inhalt TEXT,
    ->   INDEX (f_titel)
    -> );
mysql> GRANT ALL PRIVILEGES ON forum.* TO rbuser@localhost;
mysql> FLUSH PRIVILEGES;
```

Nach den Erläuterungen zum Gästebuch – und durch die vielen Kommentare – dürfte der Code des Skripts *forum.php* eigentlich ohne Weiteres verständlich sein. Beachten Sie, dass verschiedene Aufrufe dieses Skripts neben dem bereits besprochenen Modus noch andere Daten als CGI-Parameter miteinander austauschen – beispielsweise Beitrags-IDs oder die Information, ob die Threads in der Liste auf- oder zugeklappt erscheinen sollen.

Außerdem sollte die Funktion `zeige_thread()` noch erwähnt werden. Sie gibt eine Liste sämtlicher Antworten aus, die auf den Beitrag mit der angegebenen ID folgen. Dazu benutzt sie den Trick der *Rekursion*: Sie bearbeitet eine Hierarchiestufe von Antworten in einer Schleife und ruft sich für jeden einzelnen Beitrag selbst auf, um wiederum die darauf folgenden Antworten anzuzeigen. Der zweite Parameter neben der ID ist die *Stufe*, die jeweils um 1 erhöht wird, um die Threads treppenförmig einzurücken.

Die erste Anweisung dieser Funktion lautet übrigens folgendermaßen:

```
global $conn;
```

Globale (außerhalb einer Funktion definierte) Variablen können in PHP nur dann in einer Funktion verwendet werden, nachdem sie mittels `global` angekündigt wur-

den. Andernfalls würde jede Verwendung der Variablen als lokal (nur innerhalb dieser Funktion gültig) betrachtet werden.

In Beispiel 8-7 sehen Sie das vollständige Skript.

Beispiel 8-7: forum.php – ein Diskussionsforum

```
<?php

   // Include-Datei laden
   include ("util.inc.php");

   // Datenbankverbindung herstellen
   $conn = db_connect ("forum", "rbuser", "R3153n");

   // Bereits eingeloggt?
   session_start();
   $user_nummer = session_param ("user", 0);

   // Aktuellen Modus ermitteln:
   // Liste (l), Lesen (r), Eintragen (e) oder Speichern (s)?
   // Standard ist Liste
   $mode = cgi_param ("mode", "l");


   // Die Funktion zeige_thread() zeigt alle auf den
   // angegebenen Beitrag folgenden Einträge rekursiv an
   function zeige_thread ($pid, $stufe) {
      // Auf die globale Datenbankverbindung zugreifen
      global $conn;

      // Daten aller Postings mit der Parent-ID $id auslesen
      $query = $conn->query ("SELECT f_id, f_titel, f_nick FROM fo_eintraege WHERE
      f_pid=$pid ORDER BY f_id ASC");
      while (list ($id, $titel, $nick) = $query->fetch_row ()) {
         // Einrückung
         for ($i = 0; $i < $stufe; $i++) {
            echo ("  ");
         }

         // Nickname leer? -> "Anonymous"
         if ($nick == "") {
            $nick = "Anonymous";
         }

         // Titel leer? -> "[Ohne Titel]";
         if ($titel == "") {
            $titel = "[Ohne Titel]";
         }

         echo ("<a href=\"forum.php?mode=r&id=$id\">$titel</a> von $nick<br />");
```

Beispiel 8-7: forum.php – ein Diskussionsforum (Fortsetzung)

```
         // Rekursiver Aufruf für die Nachfolger von $id
         zeige_thread ($id, $stufe + 1);
      }
   }

?>

<html>
  <head>
    <title>Reiseb&uuml;ro: Forum</title>
    <link rel="stylesheet" type="text/css" href="main.css" />
<script language="JavaScript" type="text/javascript" src="utils.js"></script>
  </head>
  <body>
    <table border="0" width="750" align="center">
    <tr>
    <td>

 <br /><div align="center"><img src="logo.
gif" width="480" height="80" alt="EuroCityTravel" /></div><br />
    <!-- Einfache Navigationsleiste -->
    <!-- ... hier aus Platzgründen weggelassen ... -->
    <!-- Ende der Navigationsleiste> -->
<?php

   if ($mode == "l") {
      // Listenmodus: Eintragsliste anzeigen
      echo ("<h2>Diskussionsforum</h2>");

      // Threads auf- oder zugeklappt?
      $offen = cgi_param ("o", 1);

      // Links zum Schreiben sowie Auf- und Zuklappen
      echo ("<a href=\"forum.php?mode=e&p=0\">Neuer Beitrag</a> | ");
      if ($offen) {
         echo ("Aufklappen | <a href=\"forum.php?mode=l&o=0\">Zuklappen</a>");
      } else {
         echo ("<a href=\"forum.php?mode=l&o=1\">Aufklappen</a> | Zuklappen");
      }
      echo ("<br /><br />");

      // Thread-Startbeiträge (mit Parent-ID 0) auswählen
      $querytext = "SELECT f_id, f_titel, f_nick FROM fo_eintraege WHERE
      f_pid=0 ORDER BY f_id DESC";

      // Abfrage senden
      $query = $conn->query ($querytext);

      // Sind Beiträge vorhanden?
      $anzahl = $query->num_rows;
      if ($anzahl == 0) {
```

Beispiel 8-7: forum.php – ein Diskussionsforum (Fortsetzung)

```
         // Noch kein Beitrag da
         echo ("Bisher keine Beitr&auml;ge.");
      } else {
         // Liste anzeigen

         while (list ($id, $titel, $nick) = $query->fetch_row ()) {
            // Nickname leer? -> "Anonymous"
            if ($nick == "") {
               $nick = "Anonymous";
            }

            // Titel leer? -> "[Ohne Titel]";
            if ($titel == "") {
               $titel = "[Ohne Titel]";
            }

            // Beitrag anzeigen
            echo ("<a href=\"forum.php?mode=r&id=$id\">$titel</a> von $nick");

            // Thread anzeigen, falls aufgeklappt
            if ($offen) {
               echo ("<br /><br />");
               zeige_thread ($id, 1);
            }

            // Trennlinie
            echo ("<hr />");
         }
      }

   } elseif ($mode == "r") {
      // Lesemodus: Eintrag anzeigen

      echo ("<h2>Diskussionsforum</h2>");

      // Links für Liste Neueintrag
      echo ("<a href=\"forum.php?mode=e&p=0\">Neuer Beitrag</a> |
      <a href=\"forum.php?mode=l&o=1\">Zur &Uuml;bersicht</a><br /><br />");

      // ID lesen - Standard: -1 für nicht vorhanden
      $id = cgi_param ("id", -1);
      // Abfrage für den Beitrag
      $querytext = "SELECT DATE_FORMAT(f_datum, '%d.%m.%Y, %H:%i'),
                    f_pid, f_nick, f_mail, f_titel, f_inhalt
                    FROM fo_eintraege
                    WHERE f_id=$id";

      $query = $conn->query ($querytext);

      // Beitrag vorhanden?
      $anzahl = $query->num_rows;
      if ($anzahl == 0) {
```

Beispiel 8-7: forum.php – ein Diskussionsforum (Fortsetzung)

```
         // Nicht vorhanden
         echo ("Dieser Beitrag ist leider nicht vorhanden.<br />");
      } else {
         // Den Beitrag anzeigen
         list ($datum, $pid, $nick, $mail, $titel, $text) =
               $query->fetch_row();
         // Nickname leer? -> "Anonymous"
         if ($nick == "") {
            $nick = "Anonymous";
         }

         // Titel leer? -> "[Ohne Titel]";
         if ($titel == "") {
             $titel = "[Ohne Titel]";
         }

         // Titel ausgeben
         echo ("<h3>$titel</h3>");

         // Nickname ausgeben
         if ($mail != "") {
            // E-Mail-Adresse vorhanden? -> Nickname als Link
            echo ("von <a href=\"mailto:$mail\">$nick</a>");
         } else {
            // Nur Nickname
            echo ("von $nick");
         }

         // Datum ausgeben
         echo (", $datum<br /><br />");

         // Text ausgeben
         echo ("$text");

         // Trennlinie
         echo ("<hr />");

         // Link zum Antworten
         echo ("<a href=\"forum.php?mode=e&p=$id\">Beitrag beantworten</a>");

         // Folge-Postings ausgeben, falls vorhanden
         echo ("<h3>Bisherige Antworten</h3>");
         zeige_thread ($id, 0);
      }
   } elseif ($mode == "e") {
      // Schreibmodus: Beitragsformular anzeigen

      // Parent-ID lesen - Standard 0 (neuer Beitrag)
      $pid = cgi_param ("p", 0);

?>
```

Beispiel 8-7: forum.php – ein Diskussionsforum (Fortsetzung)

```
<h2>Forumsbeitrag verfassen</h2>
<a href="forum.php?mode=r">Eintr&auml;ge lesen</a>
<form action="forum.php" method="post">
<input type="hidden" name="mode" value="s" />
<input type="hidden" name="p" value="<?php echo ($pid); ?>" />
<table border="0" cellpadding="4">
  <tr>
    <td>Nickname:</td>
    <td><input type="text" name="nick" size="50" /></td>
  </tr>
  <tr>
    <td>E-Mail (optional):</td>
    <td><input type="text" name="mail" size="50" /></td>
  </tr>
  <tr>
    <td>Titel:</td>
    <td><input type="text" name="titel" size="50" /></td>
  </tr>
  <tr>
    <td valign="top">Ihr Eintrag:</td>
    <td><textarea name="eintr" cols="40" rows="7" wrap="virtual"></textarea></td>
  </tr>
  <tr>
    <td> </td>
    <td><input type="submit" value="Eintragen" />
  </tr>
</table>
<?php

   } elseif ($mode == "s") {
      // Speichermodus: Neuen Beitrag speichern

      // Formularfelder auslesen
      $pid = cgi_param ("p", "");
      $nick = cgi_param ("nick", "");
      $mail = cgi_param ("mail", "");
      $titel = cgi_param ("titel", "");
      $eintr = cgi_param ("eintr", "[kein Text]");

      // SQL-String für die Einfügeabfrage
      $querytext = "INSERT INTO fo_eintraege
                    (f_pid, f_datum, f_nick, f_mail, f_titel, f_inhalt)
                    VALUES (\"$pid\", NOW(), \"$nick\", \"$mail\", \"$titel\",
                    \"$eintr\")";
      // Abfrage senden
      $conn->query ($querytext);

      // Hat es geklappt?
      if ($conn->affected_rows == 1) {
         echo ("<br /><br />Ihr Beitrag wurde erfolgreich hinzugef&uuml;gt.<br />
         <br />");
```

Beispiel 8-7: forum.php – ein Diskussionsforum (Fortsetzung)

```
      } else {
         echo ("<br /><br />
               Aufgrund eines Fehlers konnte Ihr Beitrag leider nicht hinzugef&uuml;gt
               werden.<br /><br />");
      }

      echo ("<a href=\"forum.php?mode=l&o=1\">Zur &Uuml;bersicht</a>");
   } else {

?>
<h2>Fehler</h2>
Ung&uuml;ltiger Zugriff auf das Forum.<br />
<a href="forum.php?mode=l&o=1">Zur&uuml;ck</a>
<?php

   }

?>
</td>
</tr>
</table>
</body>
</html>
```

Reiseanfrage und -angebote

Die hier vorgestellten Skripten sind eine wichtige und interessante Teilanwendung der Reisebüro-Website. Nach der Auswahl von Abflug- und Zielort sowie den Reisedaten werden passende Flüge und – auf Wunsch – auch Hotelangebote herausgesucht. Die hier gezeigte Teilanwendung besteht aus zwei Skripten: *auskunft.php* dient ausschließlich der Datenauswahl; *ergebnis.php* zeigt anschließend die passenden Angebote an und enthält Formularfelder, damit eine Reise zur Buchung ausgewählt werden kann.

Reisedaten auswählen

Das Skript *auskunft.php* dient der Auswahl der gewünschten Reisedaten: Startort, Zielort, Hinreisedatum und Rückreisedatum. Außerdem kann ausgewählt werden, ob Hotelinformationen erwünscht sind oder nicht. Die Version auf der CD-ROM enthält eine interessante Erweiterung: Das Auswahlmenü für die Zielorte wird per Ajax geändert, sobald der Startort gewählt ist, damit nur die tatsächlich verfügbaren Flüge angeboten werden.

Der einzige Zugriff auf die Datenbank dient der Auswahl sämtlicher Städte – samt den zugehörigen Primärschlüsseln, die als Formulardaten übertragen werden. Die entsprechende Auswahlabfrage sieht folgendermaßen aus:

```
SELECT st_nr, st_name, la_name
FROM rb_staedte INNER JOIN rb_laender ON st_land=la_nr
ORDER BY la_name ASC, st_name ASC
```

Die ermittelten Daten werden in Arrays gespeichert – das spart einen zweiten Datenbankzugriff, da sie in zwei Formularfeldern für den Abreise- beziehungsweise Zielort erscheinen sollen. Interessant ist noch die in der Anwendung definierte PHP-Funktion `datumswahl()`. Sie erwartet einen Basisnamen als Argument und gibt drei Pull-down-Menüs für Tag, Monat und Jahr aus; diese werden durch den Basisnamen mit angehängtem Suffix (*_tag*, *_monat* beziehungsweise *_jahr*) eindeutig bezeichnet. Die Auslagerung in eine Funktion lohnt sich, weil in diesem Skript zwei Datumseingaben erforderlich sind.

In Abbildung 8-3 sehen Sie, wie das ausgefüllte Auskunftsformular im Browser aussieht.

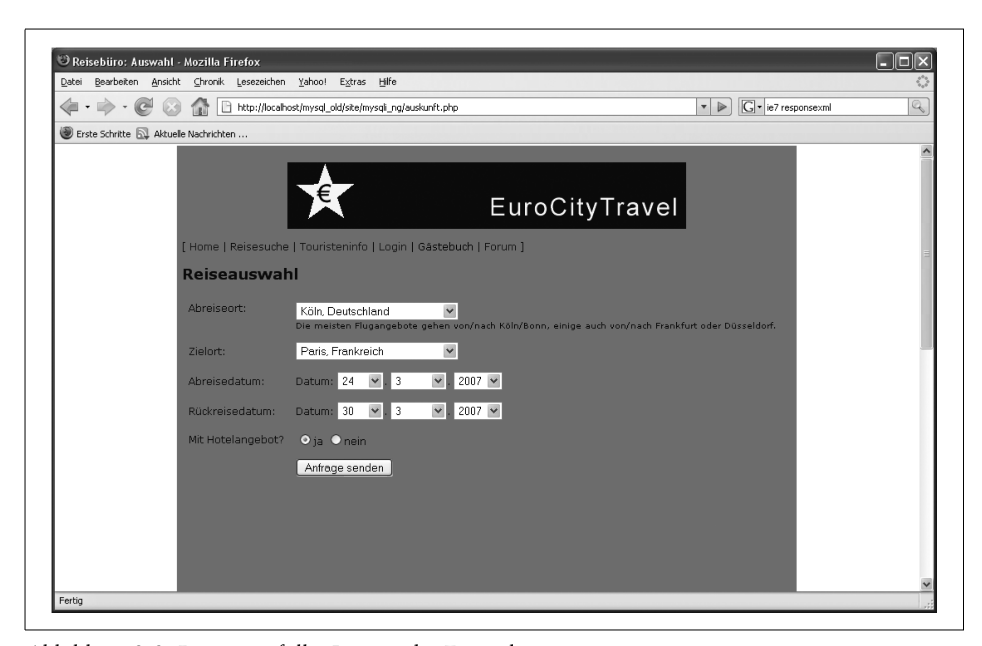


Abbildung 8-3: Das ausgefüllte Reisesuche-Formular

Beispiel 8-8 stellt den ausführlich kommentierten Quellcode dar.

Beispiel 8-8: auskunft.php – Auswahl der Reisedaten

```
<?php

   include ("util.inc.php");
   $conn = db_connect ("reisebuero", "rbuser", "R3153n");
```

Beispiel 8-8: auskunft.php – Auswahl der Reisedaten (Fortsetzung)

```
   // Funktion zur Wahl eines Datums
   function datumswahl ($feldname) {
      // Tageswähler
      echo ("<select name=\"{$feldname}_tag\" size=\"1\">\n");
      echo ("<option value=\"0\">Tag</option>\n");
      for ($i = 1; $i <= 31; $i++) {
         echo ("<option value=\"$i\">$i</option>\n");
      }
      echo ("</select>.\n");
      // Monatswähler
      echo ("<select name=\"{$feldname}_monat\" size=\"1\">\n");
      echo ("<option value=\"0\">Monat</option>\n");
      for ($i = 1; $i <= 12; $i++) {
         echo ("<option value=\"$i\">$i</option>\n");
      }
      echo ("</select>.\n");
      // Aktuelles Jahr ermitteln
      $jahr = date("Y", time());
      // Wähler für aktuelles und nachfolgendes Jahr
      echo ("<select name=\"{$feldname}_jahr\" size=\"1\">\n");
      echo ("<option value=\"$jahr\" selected=\"selected\">$jahr</option>\n");
      $jahr++;
      echo ("<option value=\"$jahr\">$jahr</option>\n</select>\n");
   }

   // Bereits eingeloggt?
   session_start();
   $user_nummer = session_param ("user", 0);

   // Liste aller Städte auslesen
   $stnr = array();
   $staedte = array();
   $querytext = "SELECT st_nr, st_name, la_name FROM rb_staedte INNER JOIN rb_laender
   ON st_land=la_nr ORDER BY la_name ASC, st_name ASC";
   $query = $conn->query ($querytext);
   while (list ($nr, $stadt, $land) = $query->fetch_row()) {
      array_push ($stnr, $nr);
      array_push ($staedte, "$stadt, $land");
   }

?>
<html>
  <head>
    <title>Reiseb&uuml;ro: Auswahl</title>
    <link rel="stylesheet" type="text/css" href="main.css" /><script
    language="JavaScript" type="text/javascript" src="utils.js"></script>
  </head>
  <body>
    <table border="0" width="750" align="center">
    <tr>
    <td>
```

Beispiel 8-8: auskunft.php – Auswahl der Reisedaten (Fortsetzung)

```
     <br /><div align="center"><img src="logo.gif" width="480" height="80"
    alt="EuroCityTravel" /></div><br />

    <!-- Einfache Navigationsleiste -->
    <!-- ... -->
    <!-- Ende der Navigationsleiste> -->

    <h2>Reiseauswahl</h2>
<?php

   // Fehler bei der vorherigen Auswahl?
   $fehler = cgi_param ("f", "");
   if ($fehler) {
      echo ("<i>Ein Fehler ist aufgetreten: $fehler</i><br />");
   }

?>
    <form action="ergebnis.php" method="post">
      <table border="0" cellpadding="4">
      <tr>
        <td valign="top">Abreiseort:</td>
        <td>
          <select name="start" size="1">
            <option value="-1">[Bitte w&auml;hlen]</option>
<?php

   for ($i = 0; $i < sizeof ($staedte); $i++) {
      $nr = $stnr [$i];
      $stadt = $staedte [$i];
      echo ("<option value=\"$nr\">$stadt</option>\n");
   }

?>
          </select><br />
          <span style="font-size: 10px">Die meisten Flugangebote gehen von/nach
          K&ouml;ln/Bonn, einige auch von/nach Frankfurt oder D&uuml;sseldorf.</span>
        </td>
      </tr>
      <tr>
        <td>Zielort:</td>
        <td>
          <select name="ziel" size="1">
            <option value="-1">[Bitte w&auml;hlen]</option>
<?php

   for ($i = 0; $i < sizeof ($staedte); $i++) {
      $nr = $stnr [$i];
      $stadt = $staedte [$i];
      echo ("<option value=\"$nr\">$stadt</option>\n");
   }

?>
```

Beispiel 8-8: auskunft.php – Auswahl der Reisedaten (Fortsetzung)

```
          </select>
        </td>
      </tr>
      <tr>
        <td>Abreisedatum:</td>
        <td>Datum: <?php datumswahl ("start"); ?></td>
      </tr>
      <tr>
        <td>R&uuml;ckreisedatum:</td>
        <td>Datum: <?php datumswahl ("ende"); ?></td>
      </tr>
      <tr>
        <td>Mit Hotelangebot?</td>
        <td>
          <input type="radio" name="hotel" value="1" checked="checked" />ja
          <input type="radio" name="hotel" value="0" />nein
        </td>
      </tr>
      <tr>
        <td> </td>
        <td><input type="submit" value="Anfrage senden" />
      </tr>
      </table>
    </form>
  </td>
  </tr>
  </table>
  </body>
</html>
```

Anzeige der Reiseangebote

Das Skript *ergebnis.php* ist umfangreicher und etwas komplexer. Zuerst liest es die von *auskunft.php* übergebenen Formulardaten aus und führt eine Reihe von Plausibilitätskontrollen durch:

- Wurde der Start- oder Zielort vergessen?
- Sind Start- und Zielort identisch?
- Liegt das gewählte Abreisedatum vor dem heutigen Datum?
- Liegt das gewählte Rückreisedatum vor dem Abreisedatum?

Wenn eine dieser Fragen mit Ja beantwortet werden muss, sendet das Skript eine entsprechende Fehlermeldung an *auskunft.php* zurück. Da der erste PHP-Block vor Beginn des HTML-Dokuments liegt, kann der entsprechende Rücksprung per `Location`-Header erfolgen (dieses Konzept wurde bereits in Kapitel 3 erläutert). *auskunft.php* überprüft, ob ihm eine Fehlermeldung übergeben wurde, und gibt diese aus.

Die ersten beiden Datenbankabfragen ermitteln anhand der Stadtnummern jeweils den Primärschlüssel, den Namen und das Kürzel des Start- beziehungsweise Zielflughafens:

```
SELECT ap_nr, ap_name, ap_kuerzel FROM rb_airports
WHERE ap_stadt=$start

SELECT ap_nr, ap_name, ap_kuerzel FROM rb_airports
WHERE ap_stadt=$ziel
```

Zunächst werden alle infrage kommenden Hinflüge aus der Tabelle *rb_flugstrecken* herausgesucht:

```
SELECT fs_nr, fs_onr FROM rb_flugstrecken
WHERE fs_start=$start_ap_nr AND fs_ziel=$ziel_ap_nr
```

Alle Ergebnisse dieser Abfrage werden nun auf die Tabelle *rb_fluege* angewendet, in der die konkreten Flugdaten stehen. Aus dieser Tabelle werden Uhrzeit und Flugpreis ermittelt, wobei das gewünschte Reisedatum als Auswahlkriterium dient.

```
SELECT fl_preis, fl_zeit FROM rb_fluege
WHERE fl_strecke=$fs_nr AND fl_datum=\"$flugdatum\"
```

Entspricht der Flug dem gewünschten Datum, wird er mit allen relevanten Informationen ausgegeben. In der vollständigen Anwendung auf der beiliegenden CD-ROM ist in der Zeile zusätzlich ein Radio-Button enthalten, mit dem die »Kunden« den jeweiligen Flug auswählen können.

Für den Rückflug werden noch einmal die gleichen Schritte unternommen – dabei werden lediglich die beiden Orte vertauscht. An dieser Stelle hätte man über eine Auslagerung der gesamten Flugsuche in eine Funktion nachdenken können, allerdings wäre dies wegen der diversen geänderten beziehungsweise vertauschten Parameter ein wenig unpraktisch.

Falls auch Hotelauskünfte ausgegeben werden sollen, werden diese mithilfe der folgenden Abfrage ausgewählt und anschließend in einer Tabelle dargestellt (auch hier müssen Sie sich die Radio-Buttons dazudenken):

```
SELECT ht_name, ht_ezpreis, ht_dzpreis, ht_bad, ht_mahlzeit
FROM rb_hotels WHERE ht_stadt=$ziel
```

In Abbildung 8-4 können Sie das Ergebnis der zuvor angeforderten Reiseauskunft sehen. Beispiel 8-9 zeigt den Quellcode des Skripts.

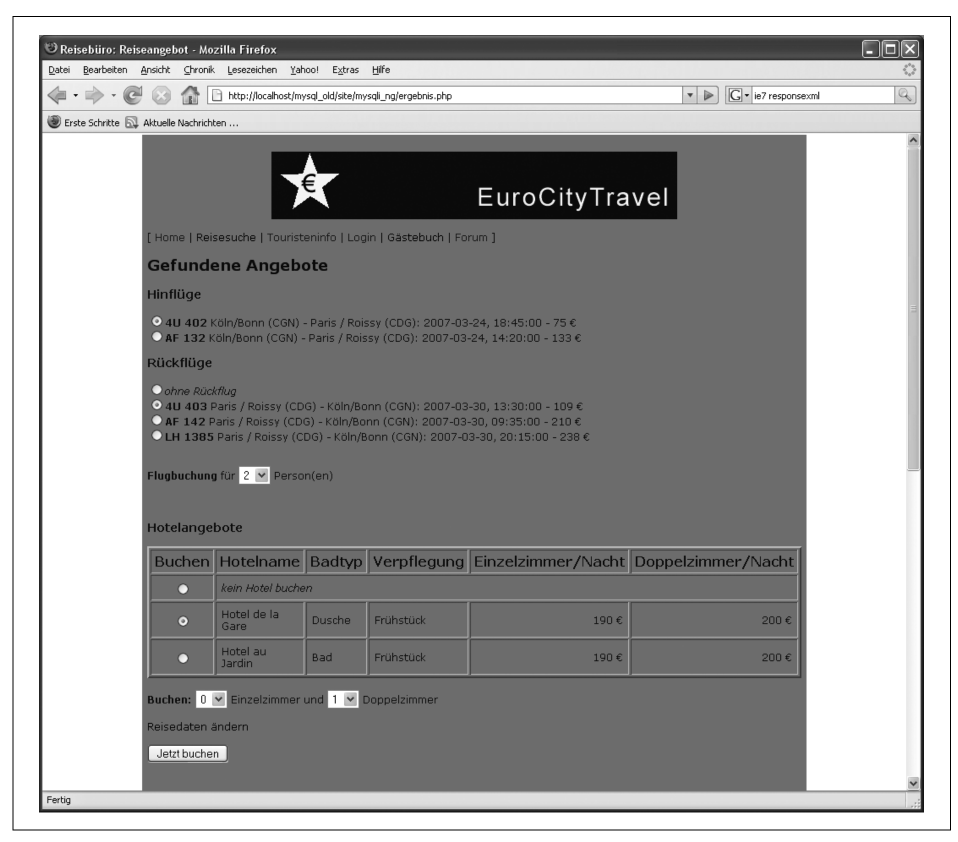


Abbildung 8-4: Anzeige der gefundenen Reiseangebote mit Buchungsformular

Beispiel 8-9: ergebnis.php – Ausgabe der Reiseangebote

```
<?php

   include ("util.inc.php");
   $conn = db_connect ("reisebuero", "rbuser", "R3153n");

   // Funktion zur Wahl eines Datums
   function datumswahl ($feldname) {
      // Tageswähler
      echo ("<select name=\"{$feldname}_tag\" size=\"1\">\n");
      echo ("<option value=\"0\">Tag</option>\n");
      for ($i = 1; $i <= 31; $i++) {
         echo ("<option value=\"$i\">$i</option>\n");
      }
      echo ("</select>.\n");
      // Monatswähler
      echo ("<select name=\"{$feldname}_monat\" size=\"1\">\n");
      echo ("<option value=\"0\">Monat</option>\n");
      for ($i = 1; $i <= 12; $i++) {
```

Beispiel 8-9: ergebnis.php – Ausgabe der Reiseangebote (Fortsetzung)

```
         echo ("<option value=\"$i\">$i</option>\n");
      }
      echo ("</select>.\n");
      // aktuelles Jahr ermitteln
      $jahr = date("Y", time());
      // Wähler für aktuelles und nachfolgendes Jahr
      echo ("<select name=\"{$feldname}_jahr\" size=\"1\">\n");
      echo ("<option value=\"$jahr\" selected=\"selected\">$jahr</option>\n");
      $jahr++;
      echo ("<option value=\"$jahr\">$jahr</option>\n</select>\n");
   }

   // Bereits eingeloggt?
   session_start();
   $user_nummer = session_param ("user", 0);

   // Liste aller Städte auslesen
   $stnr = array();
   $staedte = array();
   $querytext = "SELECT st_nr, st_name, la_name FROM rb_staedte INNER JOIN rb_
laender ON st_land=la_nr ORDER BY la_name ASC, st_name ASC";
   $query = $conn->query ($querytext);
   while (list ($nr, $stadt, $land) = $query->fetch_row()) {
      array_push ($stnr, $nr);
      array_push ($staedte, "$stadt, $land");
   }

?>
<html>
  <head>
    <title>Reiseb&uuml;ro: Auswahl</title>
    <link rel="stylesheet" type="text/css" href="main.css" />
    <script language="JavaScript" type="text/javascript" src="utils.js"></script>
  </head>
  <body>
    <table border="0" width="750" align="center">
    <tr>
    <td>
     <br /><div align="center"><img src="logo.gif" width="480" height="80"
    alt="EuroCityTravel" /></div><br />

    <!-- Einfache Navigationsleiste -->
    <!-- ... -->
    <!-- Ende der Navigationsleiste> -->

    <h2>Reiseauswahl</h2>
<?php

   // Fehler bei der vorherigen Auswahl?
   $fehler = cgi_param ("f", "");
   if ($fehler) {
```

Beispiel 8-9: ergebnis.php – Ausgabe der Reiseangebote (Fortsetzung)

```
      echo ("<i>Ein Fehler ist aufgetreten: $fehler</i><br />");
   }

?>
    <form action="ergebnis.php" method="post">
      <table border="0" cellpadding="4">
      <tr>
        <td valign="top">Abreiseort:</td>
        <td>
          <select name="start" size="1">
            <option value="-1">[Bitte w&auml;hlen]</option>
<?php

   for ($i = 0; $i < sizeof ($staedte); $i++) {
      $nr = $stnr [$i];
      $stadt = $staedte [$i];
      echo ("<option value=\"$nr\">$stadt</option>\n");
   }

?>
          </select><br />
          <span style="font-size: 10px">Die meisten Flugangebote gehen von/nach
          K&ouml;ln/Bonn, einige auch von/nach Frankfurt oder D&uuml;sseldorf.</span>
        </td>
      </tr>
      <tr>
        <td>Zielort:</td>
        <td>
          <select name="ziel" size="1">
            <option value="-1">[Bitte w&auml;hlen]</option>
<?php

   for ($i = 0; $i < sizeof ($staedte); $i++) {
      $nr = $stnr [$i];
      $stadt = $staedte [$i];
      echo ("<option value=\"$nr\">$stadt</option>\n");
   }

?>
          </select>
        </td>
      </tr>
      <tr>
        <td>Abreisedatum:</td>
        <td>Datum: <?php datumswahl ("start"); ?></td>
      </tr>
      <tr>
        <td>R&uuml;ckreisedatum:</td>
        <td>Datum: <?php datumswahl ("ende"); ?></td>
      </tr>
```

Beispiel 8-9: ergebnis.php – Ausgabe der Reiseangebote (Fortsetzung)

```
      <tr>
        <td>Mit Hotelangebot?</td>
        <td>
          <input type="radio" name="hotel" value="1" checked="checked" />ja
          <input type="radio" name="hotel" value="0" />nein
        </td>
      </tr>
      <tr>
        <td> </td>
        <td><input type="submit" value="Anfrage senden" />
      </tr>
      </table>
    </form>
  </td>
  </tr>
  </table>
  </body>
</html>
```

In diesem Kapitel:

- Benutzerverwaltung
- MySQL-Serverprogramme und -skripten
- Import und Export von Tabellendaten
- Konfigurationsdateien
- Replikation

KAPITEL 9

MySQL-Administration

Herrschen lernt sich leicht, Regieren schwer.
Johann Wolfgang von Goethe

In diesem Kapitel werden die wichtigsten Informationen zur Administration des MySQL-Servers zusammengefasst. Einiges wurde bereits in früheren Kapiteln angesprochen; hier erhalten Sie vor allem eine systematische Übersicht.

Benutzerverwaltung

Einer der wichtigsten Aspekte für die Sicherheit eines Serversystems ist eine funktionierende Benutzerverwaltung mit unterschiedlichen Berechtigungen. MySQL kennt zahlreiche unterschiedliche Benutzerrechte, die den jeweiligen Benutzern für Datenbanken, Tabellen oder sogar einzelne Spalten erteilt werden können. In diesem Abschnitt werden die MySQL-Anweisungen zur Verwaltung dieser Benutzerrechte vorgestellt, zudem wird kurz gezeigt, wie Sie sie über *phpMyAdmin* bearbeiten können.

MySQL-Anweisungen zur Benutzerverwaltung

MySQL verwendet drei Informationen zur Überprüfung eines Benutzers: Username, Passwort und den Host, von dem der User zugreift. MySQL-Benutzer und ihre Rechte haben grundsätzlich nichts mit der Benutzerverwaltung des Betriebssystems zu tun. Stattdessen werden sie insbesondere in folgenden Tabellen der Verwaltungsdatenbank *mysql* gespeichert:

- `user` – die Benutzer, ihre verschlüsselten Passwörter und ihre globalen Rechte (an allen Tabellen aller Datenbanken)
- `db` – Rechte der Benutzer an ganzen Datenbanken
- `tables_priv` – Rechte der Benutzer an einzelnen Tabellen
- `columns_priv` – Rechte der Benutzer an einzelnen Tabellen

Wenn sich ein Benutzer anmeldet, wird zunächst in der Tabelle *user* anhand der drei Aspekte Username, Passwort und Host überprüft, ob er sich beim Server anmelden darf. Anschließend werden die vier Tabellen in der hier aufgelisteten Reihenfolge durchsucht, um zu prüfen, ob eine bestimmte Operation erlaubt ist. Wenn ein allgemeineres Recht als das erforderliche gefunden wird, endet die Suche.



Das bedeutet, dass es nicht möglich ist, eine bestimmte Form des Zugriffs allgemein zu erlauben und diese dann auf einer spezielleren Ebene einzuschränken. Beispielsweise kann ein User nicht das Recht besitzen, auf eine gesamte Datenbank zuzugreifen, aber auf eine bestimmte Tabelle innerhalb dieser Datenbank nicht. Die korrekte Lösung für dieses Problem besteht darin, dem Benutzer den Zugriff auf jede relevante Tabelle einzeln, aber nicht auf die Datenbank als solche zu erlauben. Allgemein kann man sagen, dass Sie stets so wenige Rechte vergeben sollten wie möglich.

Sie müssen sich als *root* anmelden, um Benutzerrechte verwalten zu dürfen – oder mit einem Benutzernamen, dem *root* dieses Recht explizit erteilt hat.

Ein neuer Benutzer wird mithilfe der Anweisung `CREATE USER` angelegt. Die Syntax lautet wie folgt:

```
CREATE USER Benutzer[@Host] [IDENTIFIED BY "Passwort"]
```

Wenn Sie das `@`-Zeichen und den Hostnamen weglassen, darf der betreffende User von jedem beliebigen Host aus zugreifen. Dies ist aus Sicherheitsgründen aus nicht sonderlich empfehlenswert. Falls der Zugriff jedoch über das Internet erfolgen soll und die beteiligten Rechner dynamisch vom Provider vergebene IP-Adressen besitzen, geht es mitunter nicht anders. Wann immer möglich, sollten Sie den Usernamen für jeden konkreten Host, von dem er zugreifen kann, separat anlegen.

Das folgende Beispiel erstellt einen neuen Benutzer namens *rbadmin* mit dem Passwort *v3rr3153n*, der vom lokalen Rechner aus zugreifen darf:

```
CREATE USER rbadmin@localhost IDENTIFIED BY "v3rr3153n";
```

Um einen Benutzer umzubenennen, wird `RENAME USER` verwendet. Die folgende Anweisung benennt den soeben erstellten Benutzer *rbadmin* in *rbverwalter* um:

```
RENAME USER rbadmin TO rbverwalter@localhost;
```

Zu guter Letzt können Sie `DROP USER` verwenden, um einen Benutzer vollständig zu löschen, zum Beispiel:

```
DROP USER rbverwalter@localhost;
```

MySQL-Versionen vor 5.0 kennen die `CREATE USER`-Syntax nicht. Um hier einen neuen Benutzer anzulegen, wird stattdessen folgende Anweisung verwendet:

```
GRANT USAGE ON *.* TO Benutzer[@Host] [IDENTIFIED BY "Passwort"]
```

Das Benutzerrecht USAGE bedeutet, dass der betreffende User sich überhaupt am MySQL-Server anmelden darf; konkrete Berechtigungen an Datenbanken besitzt er damit noch nicht.

Um Benutzerrechte zu verwalten, wird die Anweisung GRANT verwendet. Sie erteilt dem angegebenen Benutzer eine bestimmte Berechtigung an der angegebenen Datenbank beziehungsweise Tabelle. Ihre grundlegende Syntax sieht folgendermaßen aus:

```
GRANT Recht[(Spalte)[, Recht[(Spalte)] ...]
ON [Datenbank|*].[Tabelle|*]
TO Benutzername[@Hostname]
[IDENTIFIED BY "Passwort"]
```

Das folgende Beispiel erteilt einem zuvor mittels CREATE USER angelegten Benutzer namens *newuser* alle Rechte an allen Datenbanken (dies sollten Sie nur in Ausnahmefällen in Betracht ziehen):

```
GRANT ALL PRIVILEGES ON *.* TO newuser@localhost;
```

Der Platzhalter *.* steht für alle Tabellen in allen Datenbanken. *Datenbankname*.* bezeichnet dagegen alle Tabellen der angegebenen Datenbank.

Das nächste Beispiel ermöglicht dem Benutzer *rb_mitarbeiter* das Auswählen, Ergänzen und Ändern von Informationen in der gesamten Datenbank *reisebuero*:

```
GRANT SELECT, INSERT, UPDATE ON reisebuero.*
TO rb_mitarbeiter@localhost;
```

Mit dem nachfolgenden Beispiel werden dem Benutzer *leser*, sofern er sich vom lokalen System aus anmeldet, Auswahlabfragen auf die Tabelle *rb_kunden* der Datenbank *reisebuero* gestattet:

```
GRANT SELECT ON reisebuero.rb_kunden TO leser@localhost;
```

Wenn *leser* das Recht erhalten soll, den Text in der Spalte *kd_bemerk* der Tabelle *rb_kunden* zu ändern, funktioniert dies wie folgt:

```
GRANT UPDATE(kd_bemerk) ON reisebuero.rb_kunden TO leser@locahost;
```

Tabelle 9-1 zeigt die wichtigsten Benutzerrechte für GRANT im Überblick. Einige von ihnen beziehen sich nicht auf Tabellen, sondern zum Beispiel auf Views oder Stored Procedures – das wird hier nicht näher erläutert, dürfte aber jeweils aus dem Kontext der Beschreibungen hervorgehen.

Tabelle 9-1: Die wichtigsten Benutzerrechte für GRANT- und REVOKE-Anweisungen

Berechtigung	Beschreibung
ALL [PRIVILEGES]	alle Rechte außer GRANT OPTION
USAGE	keine Rechte, nur Anmeldung
SELECT	Auswahlabfragen mit SELECT
INSERT	Einfügen von Daten mit INSERT

Tabelle 9-1: Die wichtigsten Benutzerrechte für GRANT- und REVOKE-Anweisungen (Fortsetzung)

Berechtigung	Beschreibung
UPDATE	Änderung von Daten mit UPDATE
DELETE	Löschen von Daten mit DELETE
CREATE	Tabellenerstellung mit CREATE TABLE
DROP	Löschen von Tabellen mit DROP TABLE
ALTER	Tabellenänderung mit ALTER TABLE
INDEX	Indexverwaltung mit CREATE INDEX und DROP INDEX
CREATE VIEW	Erstellung von Views mit CREATE VIEW
FILE	Import/Export mit SELECT ... INTO OUTFILE und LOAD DATA INFILE
SHOW DATABASES	Anzeigen der Datenbankliste mit SHOW DATABASES
SHOW VIEW	Anzeigen eines Views mit SHOW CREATE VIEW
CREATE ROUTINE	Erstellung von Stored Procedures/Functions
ALTER ROUTINE	Änderung von Stored Procedures/Functions
EXECUTE	Ausführen von Stored Procedures/Functions
CREATE USER	Benutzerverwaltung mit CREATE|DROP USER
SHUTDOWN	Server beenden mit mysqladmin shutdown
REPLICATION CLIENT	Ermitteln der Replikationseinstellungen
REPLICATION SLAVE	MySQL als Replikationsslave einrichten
SUPER	Serveradministration (z.B. CHANGE MASTER, KILL usw.)
GRANT OPTION	Rechteverwaltung mit GRANT/REVOKE

Die Berechtigung, GRANT selbst auszuführen, ist verständlicherweise das wichtigste und privilegierteste Benutzerrecht. Aus diesem Grund ist GRANT kein Bestandteil von ALL PRIVILEGES. Wenn Sie einem Benutzer wirklich unumschränkte Rechte an allen Datenbanken erteilen – und ihn auf diese Weise *root* gleichstellen – möchten, müssen Sie daher die Klausel WITH GRANT OPTION hinzufügen und beispielsweise Folgendes schreiben:

```
GRANT ALL PRIVILEGES TO admin@localhost WITH GRANT OPTION;
```

Um einem Benutzer ein Recht wieder zu entziehen, wird die Anweisung REVOKE mit folgender Syntax eingesetzt:

```
REVOKE Berechtigung[, Berechtigung ...] ON Objekt FROM Benutzername
```

Hier ein Beispiel, das dem Benutzer leser alle Rechte entzieht:

```
REVOKE ALL PRIVILEGES ON *.* FROM leser@localhost;
```

Beachten Sie, dass die GRANT OPTION so nicht entzogen wird – somit kann er sich beliebige Rechte selbst wieder erteilen. Die absolut sichere Variante lautet daher:

```
REVOKE ALL PRIVILEGES, GRANT OPTION ON *.* FROM leser@localhost;
```

Wenn Sie einem Benutzer alle Berechtigungen entziehen, werden seine tatsächlichen Rechte aufgehoben – um einen Benutzer ganz loszuwerden, brauchen Sie ihm also nicht jedes einzelne Recht abzuerkennen, sondern können ihn mittels `REVOKE ALL PRIVILEGES` mit einem Schlag »rechtlos« machen.

Um Passwörter zu setzen oder nachträglich zu ändern, können Sie statt der `IDENTIFIED BY`-Klausel in der `GRANT`-Anweisung auch eine `SET PASSWORD`-Anweisung verwenden. Diese hat folgende Syntax:

```
SET PASSWORD FOR Benutzername[@Hostname] =
PASSWORD|OLD_PASSWORD("Passwort")
```

Das folgende Beispiel weist dem Benutzer *newuser* ein etwas besseres Passwort als *geheim* zu:

```
SET PASSWORD FOR newuser@localhost = PASSWORD("53cr3t");
```

Die Funktionen `PASSWORD()` beziehungsweise `OLD_PASSWORD()` verschlüsseln den angegebenen Passwort-String; er wird nicht im Klartext in den `GRANT`-Tabellen gespeichert. `IDENTIFIED BY` erledigt diese Verschlüsselung dagegen automatisch.

Das Verschlüsselungsverfahren wurde ab MySQL 4.1 geändert. Ältere Client-APIs, wie sie etwa in PHP bis Version 4.3.x enthalten sind (was dann natürlich auch *phpMyAdmin* betrifft), können mit dem neuen Algorithmus nicht umgehen. Für Benutzer, die über diese APIs zugreifen, muss die Funktion `OLD_PASSWORD()` eingesetzt werden. Das folgende Beispiel erzeugt einen Benutzer namens *olduser* und weist ihm anschließend ein mithilfe der alten Methode verschlüsseltes Passwort zu:

```
CREATE USER olduser@localhost;
SET PASSWORD FOR olduser@localhost=OLD_PASSWORD("T0p53cr3t");
```

Alternativ können Sie den MySQL-Server *mysqld* auch mit der Option `--old-passwords` starten (siehe den Abschnitt »mysqld-Optionen« weiter unten). Auf diese Weise lässt sich auch `IDENTIFIED BY` zur Rechtevergabe für ältere Clients einsetzen. Empfehlenswert ist das aber nicht, da Sie aus Sicherheitsgründen das neue Verfahren verwenden sollten, sooft dies möglich ist.

Nach jeder Änderung an den Benutzerrechten sollten Sie noch folgende Anweisung ausführen, damit diese Rechte auch sofort wirksam werden:

```
FLUSH PRIVILEGES;
```

Benutzerverwaltung in phpMyAdmin

Der webbasierte Client *phpMyAdmin* ermöglicht ebenfalls die Verwaltung von Benutzerrechten – natürlich muss auch Ihr phpMyAdmin-Benutzerkonto dazu über die `GRANT`-Berechtigung verfügen. Klicken Sie zur Benutzerverwaltung auf den Link *Rechte* auf der Startseite. Zunächst gelangen Sie zu einer Übersicht über alle existierenden Benutzer und ihre bestehenden Rechte (siehe Abbildung 9-1). Um die Rechte eines Benutzers zu ändern, müssen Sie den jeweiligen Link zum Bearbeiten ganz rechts anklicken.

Unter der Liste finden Sie einen Link, um einen neuen Benutzer anzulegen. Darunter befinden sich mehrere verschieden gründliche Löschoptionen, die sich auf die zuvor angekreuzten Benutzer beziehen.

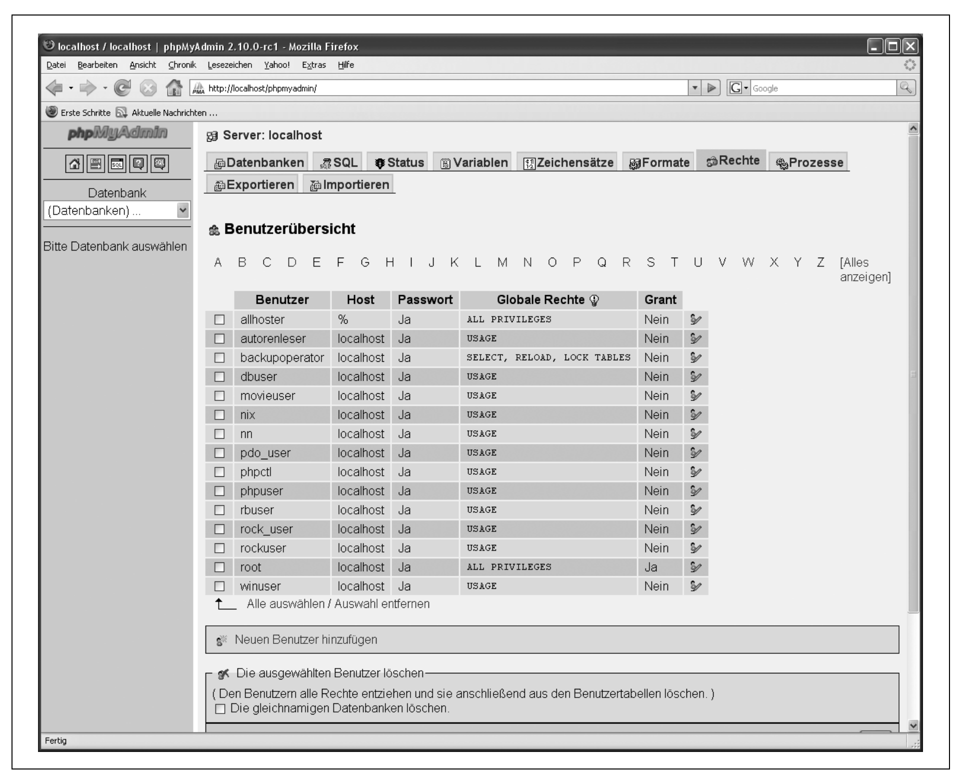


Abbildung 9-1: Startseite der Benutzerverwaltung in phpMyAdmin

Wenn Sie *Neuen Benutzer hinzufügen* wählen, wird der in Abbildung 9-2 dargestellte Dialog angezeigt. Oben werden die grundlegenden Informationen über den neuen Benutzer eingegeben: Benutzername, Host und Kennwort. Darunter können Sie detailliert die gewünschten Rechte des Benutzers auswählen; ihre Bedeutung wurde weiter oben bereits erläutert.

Die Seite zum Ändern der Rechte eines Benutzers sieht derjenigen zur Neuanlage sehr ähnlich; auch hier macht die Auswahl der globalen Berechtigungen den Hauptanteil aus. Darunter befindet sich der wichtige, nur hier vorhandene Punkt *Datenbankspezifische Rechte*, der es Ihnen ermöglicht, dem Benutzer Berechtigungen an einzelnen Datenbanken – und von der nachfolgenden Seite aus auch an einzelnen Tabellen – zu erteilen. Unter diesem Bereich finden Sie einige weitere Optionen, etwa um den Benutzernamen oder das Passwort zu ändern.

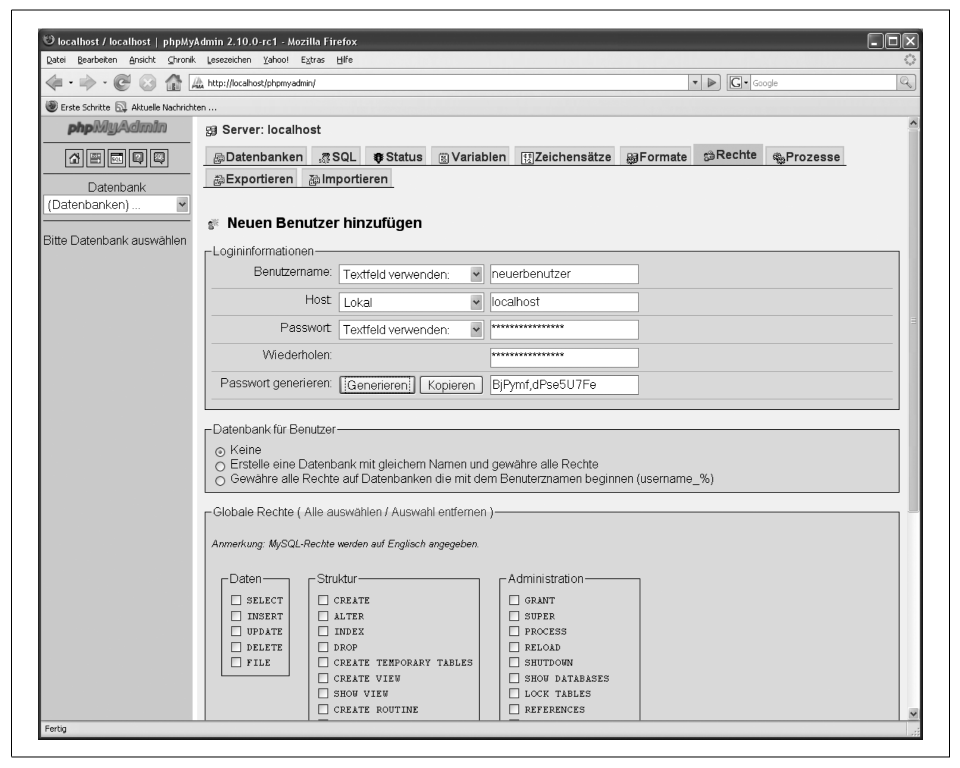


Abbildung 9-2: In phpMyAdmin einen neuen Benutzer erstellen

MySQL-Serverprogramme und -skripten

Wie im Lauf dieses Buchs bereits mehrfach erwähnt wurde, besteht der MySQL-Server nicht aus einem einzigen großen Programm, sondern aus zahlreichen einzelnen Programmen, Tools und Skripten. Einige von ihnen werden in diesem Abschnitt mit ihren wichtigsten Kommandozeilenoptionen vorgestellt.

Hier zunächst eine Übersicht über die einzelnen Serverprogramme und -skripten:

mysqld
: Standardversion des MySQL-Server-Daemons.

mysqld-max
: Erweiterte Version des MySQL-Servers, zum Beispiel mit *BerkeleyDB*- und *NDB-Cluster*-Unterstützung.

mysqld_safe
: Standard-Startskript für den MySQL-Server auf Unix-Systemen. Startet *mysqld-max*, falls vorhanden, ansonsten *mysqld*.

mysql.server

Ein weiteres Startskript, das seinerseits *mysqld_safe* aufruft. Dieses Skript kann als MySQL-Steuerskript für die verschiedenen Runlevel im *System-V-Init*-Verfahren verwendet werden, das unter Linux und einigen anderen Unix-Varianten zum Einsatz kommt (siehe den Hinweis zum automatischen Start weiter unten).

mysqld_multi

Startskript zur Steuerung mehrerer MySQL-Serverinstanzen auf demselben Host.

mysql_install_db

Skript zur Einrichtung der `GRANT`-Tabellen bei der Installation von MySQL (wurde bereits in Kapitel 2 beschrieben).

mysql_fix_privilege_tables

Skript zur Aktualisierung der `GRANT`-Tabellen nach einem MySQL-Update.

mysqlmanager

Der *MySQL Instance Manager*, ein in MySQL 5.0 neu eingeführtes Tool zur Steuerung von Serverinstanzen, auch auf entfernten Hosts. Die Beschreibung finden Sie online unter *http://dev.mysql.com/doc/mysql/en/instance-manager.html*.

Neben diesen Serverprogrammen gibt es einige weitere Tools und Skripten, die mit MySQL installiert werden:

mysqladmin

Dieses wichtige, weiter unten noch genauer besprochene Tool ermöglicht die Steuerung des laufenden MySQL-Servers, beispielsweise können Sie ihn auf diese Weise beenden oder neu starten.

myisamchk

Tool zur Überprüfung und Reparatur von *MyISAM*-Tabellen.

make_binary_distribution

Programm zur Erstellung einer MySQL-Binärdistribution aus dem selbst kompilierten Quellcode. Praktisch zur Übertragung auf weitere Rechner mit dem gleichen System. Wenn Sie eine seltenere Plattform verwenden, könnten sich andere Benutzer über einen FTP-Upload Ihrer Distribution in das Verzeichnis */pub/mysql/upload/* auf dem Server *ftp.mysql.com* freuen.

mysqlbug

Tool zur Fehlermitteilung. Alternativ können Sie MySQL-Fehler auf der Site *http://bugs.mysql.com/* eintragen.

mysqld-Optionen

Der MySQL-Server-Daemon *mysqld* besitzt zahlreiche Kommandozeilenparameter, über die sein konkretes Verhalten beim Start angepasst werden kann. Hier nur die allerwichtigsten im Überblick:

`--help [--verbose]`

Nur (ausführliche) Hilfe anzeigen und beenden.

`--ansi`

ANSI-SQL-Modus statt MySQL-Eigenheiten verwenden; wird in diesem Buch nicht behandelt. Näheres finden Sie online unter *http://dev.mysql.com/doc/mysql/en/ansi-mode.html*.

`--basedir=`*`Pfad`*

Das MySQL-Stammverzeichnis angeben; wichtig, wenn *mysqld* selbst nicht in diesem Verzeichnis liegt.

`--bind-address=`*`IP-Adresse`*

Die IP-Adresse einer Netzwerkschnittstelle; wichtig, falls mehrere vorhanden sind und der MySQL-Server nur über eine von ihnen verfügbar sein soll.

`--datadir=`*`Pfad`*

Das Datenverzeichnis, in dem sich die MySQL-Datenbanken befinden.

`--enable-named-pipe`

Clientkommunikation über Named Pipes zulassen (nur Windows NT, 2000, XP und Server 2003).

`--flush`

Änderungen nicht puffern, sondern sofort auf die Festplatte schreiben. Diese Option ist besonders sicher, aber langsamer.

`--install [`*`Dienstname`*`]`

Den MySQL-Server als automatisch startenden Dienst installieren (nur Windows).

`--log[=`*`Datei`*`]`

Verbindungen und Abfragen in eine Logdatei protokollieren.

`--log-error[=`*`Datei`*`]`

Fehler in eine Logdatei protokollieren.

`--old-passwords`

Passwörter generell im alten Verschlüsselungsformat (MySQL bis 4.0) generieren. Wenn Sie auf alte Passwörter angewiesen sind, ist das praktischer als die Verwendung der Funktion `OLD_PASSWORD()`. Allerdings sollten Sie, wann immer möglich, das neue Verschlüsselungsverfahren einsetzen, da es sicherer ist.

`--open-files-limit=`*`Anzahl`*

Die maximale Anzahl von Dateien, die *mysqld* gleichzeitig offen halten darf.

`--pid-file=`*`Pfad`*

Die Prozess-ID-Datei für den MySQL-Server.

`--port=`*`Portnummer`*

Der TCP-Port, an dem der MySQL-Server lauscht (Standard 3306).

`--remove [`*`Dienstname`*`]`

Den MySQL-Dienst entfernen (nur Windows).

`--socket=`*`Pfad`*

Unix-Domain-Socket für lokale Clientverbindungen.

`--standalone`

mysqld als Programm, aber nicht als Dienst ausführen (nur Windows).

`--user=`*`Benutzername`*

Die Unix-Benutzerkennung, unter der *mysqld* ausgeführt werden soll – in der Regel *mysql* (siehe Installationsanleitung in Kapitel 2).

`--version`

Nur Versionsinformation anzeigen und beenden.

mysqld_safe-Optionen

Wie bereits erwähnt, wird der MySQL-Server unter Unix in der Regel über das Skript *mysqld_safe* gestartet. Es aktiviert je nach Verfügbarkeit *mysqld-max* oder *mysqld*. Es besitzt diverse eigene Optionen; dies sind die wichtigsten:

`--help`

Nur Hilfe anzeigen und beenden (verfügbar seit MySQL 5.0.3).

`--basedir=`*`Pfad`*

Das Stammverzeichnis des MySQL-Servers.

`--core-file-size=`*`Byte`*

Größe der Speicherpartition, die für MySQL reserviert werden soll.

`--datadir=`*`Pfad`*

Datenverzeichnis, in dem sich die MySQL-Datenbanken befinden.

`--defaults-extra-file=`*`Pfad`*

Konfigurationsdatei, die *zusätzlich* zu den Standardoptionen verarbeitet wird.

`--defaults-file=`*`Pfad`*

Konfigurationsdatei, die *anstelle* der Standardoptionen verarbeitet wird.

`--ledir=`*`Pfad`*

Verzeichnis, in dem sich das Serverprogramm *mysqld* befindet.

`--log-error=`*`Pfad`*

Pfad der Logdatei für Fehler.

`--mysqld=`*`Programm`*

Dateiname des Serverprogramms (muss angegeben werden, wenn Server- und Datenverzeichnis verschieden sind).

`--nice=`*`Priorität`*

nice-Prioritätswert für den MySQL-Server festlegen.

`--no-defaults`

Keine Konfigurationsdateien einlesen.

`--open-files-limit=`*`Anzahl`*

Maximale Anzahl von Dateien, die der Server gleichzeitig offen halten darf.

`--pid-file=`*`Pfad`*

Die Prozess-ID-Datei für den MySQL-Server.

`--port=`*`Portnummer`*

TCP-Port des MySQL-Servers (Standard 3306).

`--socket=`*`Pfad`*

Unix-Domain-Socket für lokale Clientverbindungen.

`--timezone=`*`Zone`*

Angabe der Standard-Zeitzone.

`--user=`*`Benutzername`*

Benutzerkennung, unter der der Server ausgeführt werden soll – in der Regel *mysql*.



Für Linux- und andere Unix-Systeme gibt es zwei verschiedene Methoden, um Daemons (Serverdienste) beim Booten automatisch zu starten: Auf dem klassischen AT&T-Unix aufsetzende Varianten und Linux verwenden das sogenannte *System V Init* mit verschiedenen Runlevels, während BSD-basierte Versionen ein eigenes Verfahren einsetzen.

Mit anderen Worten: Wenn Sie den MySQL-Server automatisch starten möchten, müssen Sie herausfinden, welches Verfahren Ihr System verwendet, und ein geeignetes Skript – beispielsweise das oben erwähnte *mysqld.server* – in ein für Startskripten geeignetes Verzeichnis (zum Beispiel */etc/rc.d*) kopieren.

Viele Distributionen verfügen übrigens jeweils über ein eigenes bequemes Verfahren, um Autostarts einzurichten – SUSE Linux enthält beispielsweise den sogenannten *Runlevel-Editor*. Näheres entnehmen Sie bitte der Dokumentation Ihres Systems.

Das Hilfsprogramm mysqladmin

Mithilfe des Programms *mysqladmin* können Sie den MySQL-Server steuern. Um dieses Programm auszuführen, müssen Sie wie beim Kommandozeilenclient mysql die Optionen `-u` *`Benutzername`* und `-p` (Passworteingabe) verwenden. Die Aufrufsyntax lautet also:

```
mysqladmin Befehl -u Benutzername -p
```

Daraufhin wird das Passwort abgefragt; falls es korrekt ist und der angegebene Benutzer Administratorrechte besitzt, wird der gewünschte Befehl ausgeführt.

Hier die wichtigsten Administrationsbefehle im Überblick:

`create` *Datenbank*

Erstellt die angegebene Datenbank neu (wie eine `CREATE DATABASE`-Abfrage).

`drop` *Datenbank*

Löscht die angegebene Datenbank (wie `DROP DATABASE`).

`extended-status`

Statusinformationen ausgeben. Sie sollten diesen Befehl zum Durchblättern mit `|less` (unter Windows `|more`) aufrufen, da die Ausgabe viele Bildschirme lang ist.

`ping`

Überprüft, ob der Server aktiv ist.

`reload`

Lädt die `GRANT`-Tabellen neu (nach der Änderung von Benutzerrechten).

`shutdown`

Beendet den MySQL-Server.

`status`

Gibt eine kurze Statusmeldung aus.

`version`

Gibt die Serverversion aus.

Das folgende Beispiel beendet den MySQL-Server:

```
$ mysqladmin shutdown -u root -p
Enter password:
```

Import und Export von Tabellendaten

Eine wichtige Fähigkeit von MySQL wurde in diesem Buch bisher noch nicht erwähnt: der Austausch von Daten mit anderen Anwendungen und Formaten, also der Import und Export von Tabellendaten. Hier werden einige grundlegende Möglichkeiten besprochen.

Das mit MySQL gelieferte Hilfsprogramm *mysqldump* ermöglicht die Speicherung von Datenbank- und Tabellendaten als Textdateien mit SQL-Anweisungen. Diese können Sie später in denselben oder einen anderen MySQL-Server (bedingt auch in andere SQL-Datenbankserver) laden. Beispielsweise wurde die Datei *reisebuero.sql* zur Erstellung der Datenbank *reisebuero* mithilfe von *mysqldump* erzeugt.

Es gibt drei grundlegende Aufrufvarianten für *mysqldump*. Die erste exportiert eine, mehrere oder alle Tabellen einer Datenbank:

```
mysqldump [Optionen] Datenbank [Tabelle ...]
```

Die nächste Methode exportiert eine oder mehrere Datenbanken:

```
mysqldump [Optionen] --databases Datenbank1 [Datenbank2 ...]
```

Die letzte schließlich exportiert alle Tabellen aus allen Datenbanken:

```
mysqldump [Optionen] --all-databases
```

Übrigens wird auch *mysqldump* normalerweise mit der bekannten Parameterkombination `-u` *`Benutzername`* `-p` aufgerufen.

Das folgende Beispiel exportiert die Datenbank *reisebuero* mit den Standardoptionen:

```
> mysqldump -u root -p reisebuero
Enter password:
```

Diese Anweisung schreibt die SQL-Anweisungen zur Erstellung der Datenbank auf die Konsole. In der Regel ist das natürlich nicht erwünscht; deshalb wird die Ausgabe üblicherweise in eine Datei umgeleitet, zum Beispiel:

```
> mysqldump -u root -p reisebuero >reisebuero.sql
```

Hier die wichtigsten Optionen von *mysqldump* (neben `-u` und `-p`) im Überblick:

`--help`
: Hilfe ausgeben und beenden.

`--add-drop-table`
: Vor jeder `CREATE TABLE`-Anweisung vorsichtshalber `DROP TABLE` einfügen, um eventuell vorhandene Tabellen zunächst zu löschen.

`--compact`
: Möglichst kurze Ausgabe, zum Beispiel ohne Kommentare.

`--compatible=`*`Version`*
: Kompatibilität mit anderen SQL-Versionen oder Datenbanken herstellen. *`Version`* kann zum Beispiel die Werte `ansi` (ANSI-SQL), `mysql323` (MySQL 3.23), `mysql40` (MySQL 4.0), `postgresql` (PostgreSQL), `oracle` (Oracle), `mssql` (Microsoft SQL Server) oder `db2` (IBM DB2) annehmen.

`--where='Kriterium'`
: Nur diejenigen Datensätze einfügen, die dem angegebenen Kriterium entsprechen.

Interessant ist zudem die Option `--xml`: Sie gibt die Datenbank und ihre Tabellen nicht im SQL-Format, sondern als wohlgeformtes XML-Dokument aus. Das eröffnet völlig neue Weiterverarbeitungsmöglichkeiten durch die zahlreichen XML-APIs moderner Programmiersprachen. (Näheres über das XML-Format und seine Anwendung erfahren Sie beispielsweise in der Onlinefassung des XML-Kapitels meines Buchs *Kompendium der Informationstechnik* unter *http://www.galileocomputing.de/openbook/kit/itkomp15000.htm*).

Daten, die mittels *mysqldump* als SQL exportiert wurden, lassen sich mithilfe der Anweisung `SOURCE` im Kommandozeilenclient mysql einlesen, zum Beispiel:

```
mysql> \. reisebuero.sql
```



Mit zu den wichtigsten Aufgaben des Exports mit *mysqldump* und des späteren Imports mithilfe der MySQL-Anweisung `SOURCE` gehören Datensicherung und -wiederherstellung (Backup und Restore). Sie sollten Ihre Datenbanken und Tabellen regelmäßig auf externe Datenträger sichern, um Datenverlusten – etwa durch Hardwareschäden, Virenbefall oder Crackerangriffe – vorzubeugen. Dabei können Sie im Prinzip genau so vorgehen wie beschrieben. In der Praxis benötigen Sie vor der Datensicherung allerdings zusätzliche `LOCK`- und `FLUSH`-Anweisungen, um die Tabellen vorübergehend zu sperren beziehungsweise im Arbeitsspeicher befindliche Daten auf die Datenträger zu schreiben. Näheres zu diesem Thema finden Sie in Abschnitt 5.8.1, *Database Backups*, in der MySQL-Onlinedokumentation (*http://dev.mysql.com/doc/mysql/en/backup.html*).

MySQL kann aber nicht nur SQL-Dateien einlesen, sondern auch Dateien mit strukturierten Daten. Dazu wird die SQL-Anweisung `LOAD DATA INFILE` verwendet. Ihre grundlegende Syntax sieht folgendermaßen aus:

```
LOAD DATA INFILE 'Dateiname'
INTO TABLE Tabellenname
[FIELDS [TERMINATED BY '...'] [ENCLOSED BY '...']]
[LINES [STARTING BY '...'] [TERMINATED BY '...']]
```

Die optionale `FIELDS`-Klausel gibt an, durch welches Zeichen die einzelnen Felder getrennt (`TERMINATED`) beziehungsweise eingeschlossen (`ENCLOSED`) werden. Standardmäßig dient der Tabulator (`\t`) als Trennzeichen.

Mithilfe von `LINES` können Sie angeben, womit die Zeilen beginnen (`STARTING`) und wodurch sie abgeschlossen werden (`TERMINATED`). Die Vorgabe ist `'\n'` (Zeilenumbruch) als Zeilenende.

Das folgende Beispiel importiert den Inhalt einer Datei *data.txt* (die das angegebene Standardformat besitzt) in die Tabelle *test*:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE test;
```

Hier ein weiteres Beispiel – es lädt die Daten aus der Datei *csv.txt* in die Tabelle *test2*; die Felder sind hier durch Kommata getrennt:

```
mysql> LOAD DATA INFILE 'csv.txt' INTO TABLE test2
    -> FIELDS TERMINATED BY ',';
```

Das Gegenstück zu `LOAD DATA INFILE` ist `SELECT ... INTO OUTFILE`. Die allgemeine Syntax für `SELECT` wurde bereits ausführlich behandelt; die Optionen für `INTO OUTFILE` entsprechen denjenigen von `LOAD DATA INFILE`. Das folgende Beispiel exportiert alle Datensätze der *reisebuero*-Tabelle *rb_airports* mit den Standardoptionen als Datei namens *airports.txt*:

```
SELECT * FROM rb_airports
INTO OUTFILE 'airports.txt';
```

Das nächste Beispiel exportiert die Daten aus *rb_airlines*, trennt sie aber durch Kommata statt durch Tabulatoren:

```
SELECT * FROM rb_airlines
INTO OUTFILE 'airlines.txt'
FIELDS TERMINATED BY ',';
```

Solche *CSV*-Dateien (*Comma Separated Values*, also durch Kommata getrennte Werte) werden zum Beispiel von Tabellenkalkulationsprogrammen wie Microsoft Excel unterstützt.

Erweiterte Exportoptionen stehen in *phpMyAdmin* unter dem Link *Exportieren* zur Verfügung. In Abbildung 9-3 wird die entsprechende Seite angezeigt. Sie fasst einige Optionen von *mysqldump* und `WRITE DATA INTO OUTFILE` zusammen.

Links können Sie zunächst die gewünschten Tabellen der aktuellen Datenbank sowie eines der folgenden Exportformate auswählen:

- *SQL* (in der Abbildung zu sehen) exportiert die gewünschten Tabellen im SQL-Format und entspricht damit im Wesentlichen den weiter oben erläuterten Optionen von *mysqldump*.
- *LaTeX* exportiert die Daten als Quelldokument in der Textsatzsprache LaTeX. Mithilfe des gleichnamigen Satzprozessors lassen sich daraus sehr hochwertig formatierte PostScript- oder PDF-Dokumente generieren.
- *CSV-Daten für MS Excel* exportiert die Daten im CSV-Format mit Besonderheiten, die das verbreitete Tabellenkalkulationsprogramm Microsoft Excel benötigt.
- *CSV-Daten* produziert ebenfalls CSV-Dateien, allerdings in einer allgemeineren Version, die von einigen anderen Programmen importiert werden kann.
- *XML* speichert die Datenbank und ihre Tabellen als XML-Dokument, genau wie die bereits erwähnte *mysqldump*-Option `--xml`.

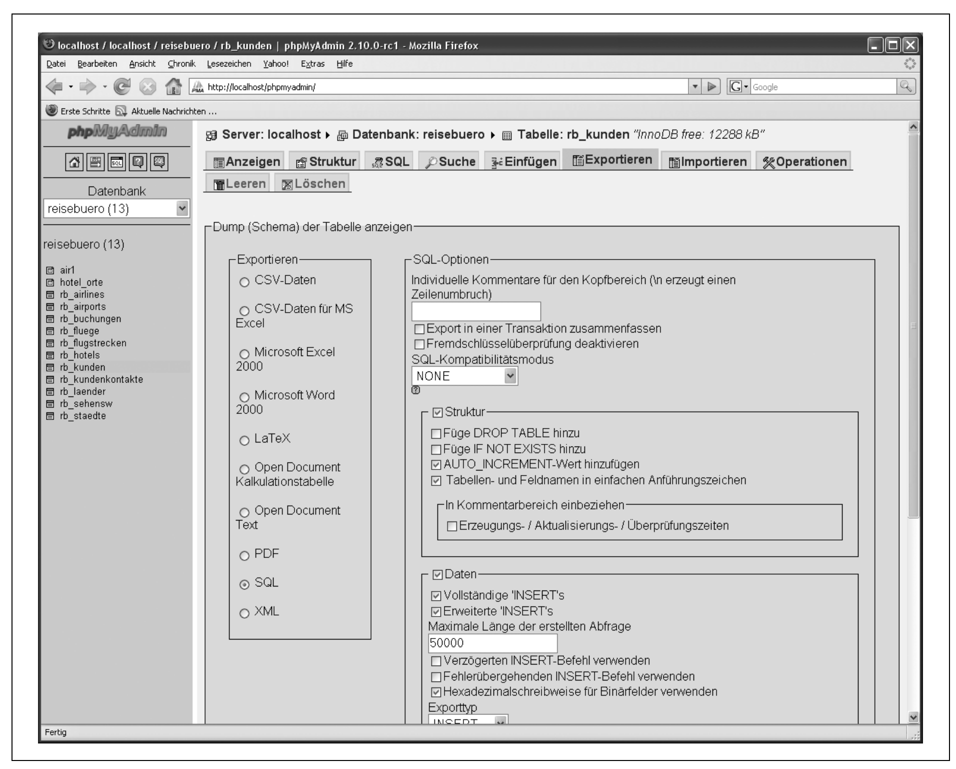


Abbildung 9-3: Der Export-Dialog in phpMyAdmin, hier für eine einzelne Tabelle

Je nach gewähltem Format werden rechts unterschiedliche Optionen angezeigt. Die Einstellungen für SQL entsprechen im Wesentlichen den *mysqldump*-Parametern; diejenigen der anderen Formate sollten Sie bei Gelegenheit ausprobieren. Das Format XML besitzt gar keine Optionen.

Konfigurationsdateien

Bereits in Kapitel 2 wurde darauf hingewiesen, dass MySQL grundsätzlich auch ohne Konfigurationsdateien funktionieren kann. Wenn Sie welche erstellen möchten, um bestimmte Standardeinstellungen zu ändern, können Sie auf einem Unix-System die beiden Dateien */etc/my.cnf* (allgemeine Konfigurationsdatei für den Server, Clients und Dienstprogramme) sowie *.my.cnf* im Home-Verzeichnis eines Benutzers (Konfigurationsdatei dieses Users; nur für Clients und Dienstprogramme) anlegen. Auf einem Windows-Rechner gibt es dagegen eine Datei namens *my.ini* im MySQL-Verzeichnis.

In MySQL-Konfigurationsdateien gibt es verschiedene Abschnitte, deren Namen die jeweiligen Einzelprogramme repräsentieren. Sie werden in eckige Klammern

gesetzt. Jede Konfigurationsanweisung steht darunter in einer eigenen Zeile und hat das Format `Parameter = Wert`. Zeilen, die mit einer Raute (#) beginnen, leiten einen Kommentar ein.

Für den MySQL-Server wird der Abschnitt `[mysqld]` verwendet. Andere wichtige Abschnitte sind `[mysql]` für den Kommandozeilenclient, `[mysqladmin]` für das gleichnamige Administrationsprogramm und `[mysqldump]` für das Exporttool.

Für den Server selbst können Sie unter anderem folgende Parameter konfigurieren:

`port =` *Port-Nummer*

TCP-Port, an dem der Server auf Verbindungen wartet. Wenn Sie mehrere MySQL-Server auf einem Host verwenden möchten, müssen Sie jedem von ihnen eine andere Portnummer zuweisen; der Standardwert ist 3306.

`socket =` *Dateipfad*

Unix-Domain-Socket für die Kommunikation zwischen Server und Clients auf dem lokalen Rechner; nur auf Unix-Systemen verfügbar. Standard ist */tmp/mysql.sock*.

`character-set-server =` *Zeichensatz*

Standard-Zeichensatz für den Server; wenn Sie nichts anderes angeben, `latin1`.

`collation-server =` *Kollation*

Standard-Sortierreihenfolge des Servers. Da MySQL aus Schweden stammt, ist der Standardwert `latin1_swedish_ci`.

`language =` *Sprache*

Sprache für Fehlermeldungen und Warnungen; Beispiele: `english` (Standard), `german`, `french`.

`sql-mode =` *Modus*

SQL-Modus des Servers, der eine gewisse Kompatibilität zu verschiedenen MySQL-Versionen beziehungsweise anderen Datenbanken ermöglicht. Standard ist `mysql`. Ein anderer wichtiger Wert ist `ansi`; er nähert die Syntax dem ANSI-SQL-Standard an. Dies erfordert zum Beispiel zwingend 'einfache Anführungszeichen' für Strings und "doppelte" für Tabellen- und Spaltennamen, die mit SQL-Schlüsselwörtern verwechselt werden könnten.

Weitere wichtige Einträge in den Konfigurationsdateien betreffen das Logging. MySQL bietet diverse Logdateien an, die der Wiederherstellung nach Datenverlusten, der Suche nach Fehlern oder der Replikation (siehe unten) dienen.

Normalerweise wird nur die Error-Logdatei geführt. Sie befindet sich im MySQL-Datenverzeichnis. Ihr Name ist der Hostname mit der Endung *.log*, etwa *dbhost.log*. Im Abschnitt `[mysqld]` der Hauptkonfigurationsdatei kann ihr Speicherort geändert werden:

```
log-error = Pfad
```

Optional, aber recht häufig genutzt, ist die binäre *Update-Logdatei*, die sämtliche Änderungsabfragen protokolliert. Sie wird unter anderem für die Replikation benötigt, kann aber auch zur Wiederherstellung nach einem Datenverlust verwendet werden. Tragen Sie eine Zeile wie diese in die Konfigurationsdatei ein, wenn Sie diese Datei benötigen:

```
[mysqld]
...
log-bin = binlog
```

Daraufhin werden im Datenverzeichnis Ihres MySQL-Servers die Dateien *binlog.000001* und *binlog.index* erzeugt. Von Zeit zu Zeit beginnt MySQL eine neue Logdatei, die dann die nächste laufende Nummer erhält – zum Beispiel *binlog.000002*.

Wenn Sie manuell neue Logdateien beginnen möchten, können Sie Folgendes eingeben:

```
mysql> FLUSH LOGS;
```

Dasselbe lässt sich auch mittels *mysqladmin* bewerkstelligen:

```
> mysqladmin flush-logs -u root -p
```

Da die Update-Logdatei binär ist, können Sie sie übrigens nicht per Texteditor lesen, sondern nur mithilfe des Tools *mysqlbinlog*. Dies geht sehr einfach folgendermaßen:

```
> mysqlbinlog Logdatei
```

Eine weitere interessante Logdatei, die vor allem der Optimierung Ihrer Abfragen dienen kann, protokolliert zu langsame Abfragen unter *Hostname-slow.log*. Tragen Sie dazu Folgendes unter den Abschnitt `[mysqld]` der Konfigurationsdatei ein:

```
log-slow-queries
long_query_time = Sekunden
```

Auch alle Abfragen, die keine Indizes verwenden, können in derselben Datei protokolliert werden, wenn Sie folgende Zeile hinzufügen:

```
log-queries-not-using-indexes
```

MySQL enthält ab Werk einige Konfigurationsdateivorlagen. Sie befinden sich unter Unix im Verzeichnis *support-files*, auf Windows-Rechnern direkt im MySQL-Verzeichnis. Sie reichen von *my-small.cnf* beziehungsweise *my-small.ini* für kleine Datenbanken bis zu *my-huge.cnf* beziehungsweise *my-huge.ini* für sehr große Datenbanken. Wenn Sie möchten, können Sie eine dieser Dateien an Ihren Server anpassen. Auf einem Unix-Rechner wird sie dann nach */etc/my.cnf* kopiert, unter Windows dagegen einfach an Ort und Stelle in *my.ini* umbenannt.

Replikation

Replikation ist das automatische Kopieren aller Änderungen von einem MySQL-Server (*Master*) auf einen oder mehrere andere (Slaves). Damit erhalten Sie automatische Backups, zudem können Sie Auswahlabfragen bei leselastigen Anwendungen auf mehrere Slaves verteilen (Load-Balancing).

Wenn Sie die Replikation erst einmal in Betrieb genommen haben, wird sie automatisch ausgeführt. Nach einem Ausfall aktualisiert sich ein Slave selbstständig.

Auf dem Master muss zuerst ein Benutzer für den Replikationsslave erzeugt werden:

```
mysql> CREATE USER replicant@Slave
    -> IDENTIFIED BY "ReplPasswort";
```

Dieser User benötigt die globale Berechtigung `REPLICATION SLAVE`:

```
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO replicant@Slave;
```

Falls noch keine Update-Logdatei besteht, müssen Sie diese nun einrichten. Anschließend müssen Sie sich den Namen der aktuellen Logdatei und ihre Position merken. Diese ermitteln Sie wie folgt:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Die benötigten Werte sind *File* (etwa *binlog.000006*) und *Position* (zum Beispiel *951*).

Als Nächstes wird der MySQL-Server auf dem Master gestoppt:

```
> mysqladmin shutdown -u root -p
```

Nun muss das gesamte Datenverzeichnis auf den Slave-Host kopiert werden. Dies funktioniert je nach Betriebssystem und Netzwerkkonfiguration unterschiedlich. Auf dem Slave müssen Sie dazu ebenfalls den MySQL-Server beenden und den Inhalt seines Datenverzeichnisses löschen. Wenn Sie seine eigenen Datenbanken später noch brauchen, müssen Sie sie zuerst in ein anderes Verzeichnis kopieren.

Tragen Sie nun folgende Zeile in die Hauptkonfigurationsdatei des Master-Servers ein:

```
[mysqld]
...
server-id = 1
```

Welche konkrete Nummer Sie wählen, ist egal; sie muss sich nur von den IDs der Slaves unterscheiden. 1 ist allerdings Standard für den Master.

Nach diesen Schritten können Sie den Master-MySQL-Server wieder starten.

Wenn Ihr Slave ein Unix-Rechner ist, müssen Sie die Berechtigungen für das kopierte Datenverzeichnis ändern:

```
# chown -R mysql:mysql MySQL-Daten
```

Auch der Slave braucht nun eine eindeutige Server-ID. Beispiel:

```
[mysqld]
...
server-id = 2
```

Starten Sie den Slave-MySQL-Server nun wieder. Öffnen Sie den MySQL-Kommandozeilenclient und geben Sie Folgendes ein (selbstverständlich müssen Sie statt der hier gezeigten Platzhalter konkrete Werte einfügen):

```
mysql> CHANGE MASTER TO
    -> MASTER_HOST =     'Master',
    -> MASTER_USER =     'replicant',
    -> MASTER_PASSWORD = 'ReplPasswort',
    -> MASTER_LOG_FILE = 'Master-Logdatei'
    -> MASTER_LOG_POS =  Position;
mysql> START SLAVE;
```

Führen Sie auf dem Master nun eine beliebige Änderungsabfrage durch. Überprüfen Sie anschließend per Auswahlabfrage auf dem Slave, ob die Änderung korrekt repliziert wurde.

Hier zum Schluss ein kleines PHP-Beispiel für Load-Balancing. Es geht davon aus, dass es einen Master und zwei Slaves gibt, und stellt eine *mysqli*-Verbindung für Änderungsabfragen mit dem Master her, während zum Lesen eine Zufallsauswahl zwischen den beiden Slaves stattfindet:

```
<?php

  // Host für Schreibzugriffe
  $schreib_host = "192.168.0.2";
  // verfügbare Hosts für Lesezugriffe
  $lese_hosts = array ("192.168.0.3", "192.168.0.4");
  // einen Lese-Host zufällig auswählen
  $lese_host = $lese_hosts[rand (0,1)];

  // weitere Verbindungsparameter
  $user = "rbuser";
  $pass = "R3153n";
  $db = "reisebuero";

  // Schreibverbindung herstellen
  $schreib_conn = new mysqli ($schreib_host, $user, $pass, $db);
  // Leseverbindung herstellen
  $lese_conn = new mysqli ($lese_host, $user, $pass, $db);

  /* Nun können beliebige Datenbankoperationen folgen.
     Wichtig: Schreibvorgänge NUR über $schreib_conn,
     Leseoperationen über $lese_conn.                  */

?>
```

In diesem Anhang:

- MySQL-Abfragen
- MySQL-Funktionen in PHP

ANHANG A

Kurzreferenz

Die Seele jeder Ordnung ist ein großer Papierkorb.
Kurt Tucholsky

Im Folgenden finden Sie eine kurze Übersicht über wichtige MySQL-Abfragen und PHP-Anweisungen zur Kommunikation mit MySQL-Datenbanken. Wie in Syntaxschemata üblich, stehen kursiv dargestellte Teile für Elemente, die Sie durch etwas Konkretes ersetzen müssen, während eckige Klammern optionale Teile einer Anweisung bezeichnen. Ein vertikaler Strich (|) steht dagegen zwischen Elementen, von denen eines ausgewählt werden soll.

MySQL-Abfragen

In diesem Abschnitt finden Sie Syntaxschemata der wichtigsten MySQL-Anweisungen und -Funktionen; sie alle werden in den verschiedenen Kapiteln dieses Buchs näher erläutert. Fortgeschrittene SQL-Möglichkeiten wie Fremdschlüssel-Constraints, Transaktionen oder Stored Procedures werden in diesem Anhang nicht behandelt; lesen Sie dazu bitte die entsprechenden Abschnitte in den Kapiteln 5 und 6.

Datenbanken erstellen

```
CREATE DATABASE Datenbankname
[[DEFAULT] CHARACTER SET Zeichensatz]
[COLLATE Sortierreihenfolge]
```

Liste aller Zeichensätze

```
SHOW CHARACTER SET
```

Liste aller Kollationen

```
SHOW COLLATION
```

Tabellen erstellen

Grundform

```
CREATE TABLE Tabellenname (
     Spaltenname1 Datentyp [Optionen]
  [, Spaltenname2 Datentyp [Optionen]
     ...]
  [, PRIMARY KEY [Indexname] (Spaltenname[, ...])]
  [, [UNIQUE|FULLTEXT] INDEX [Indexname] (Spaltenname[, ...]) ...]
)
[[DEFAULT] CHARACTER SET Zeichensatz]
[COLLATE Sortierreihenfolge]
[ENGINE=Tabellentyp]
```

Kopie der Tabellenstruktur

```
CREATE TABLE Tabellenname LIKE [Datenbank.]Tabellenname
```

Kopie des Tabelleninhalts

```
CREATE TABLE Tabellenname SELECT Auswahlkriterien
```

Die wichtigsten Tabellentypen

`MyISAM` (schneller) und `InnoDB` (Unterstützung für Transaktionen und weitere moderne Features).

Datentypen

- Ganzzahlige: `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, `BIGINT`
- Fließkomma: `FLOAT`, `DOUBLE` (Synonym: `REAL`)
- Festkomma: `DECIMAL(`*`Stellen, Nachkommastellen`*`)`
- Datum/Uhrzeit: `DATETIME`, `DATE`, `TIME`, `YEAR`, `TIMESTAMP`
- Strings: `CHAR(`*`n`*`)`, `VARCHAR(`*`n`*`)`
- Textblöcke: `TINYTEXT`, `TEXT`, `MEDIUMTEXT`, `LONGTEXT`
- Binärblöcke: `TINYBLOB`, `BLOB`, `MEDIUMBLOB`, `LONGBLOG`
- Aufzählungstypen: `ENUM(`*`Wert1,Wert2, ...`*`)`, `SET(`*`Wert1,Wert2, ...`*`)`

Wichtige Feldoptionen

- `AUTO_INCREMENT` – automatische Nummerierung
- `[NOT] NULL` – Feld darf [nicht] leer sein
- `UNSIGNED` – vorzeichenlos
- `ZEROFILL` – führende Stellen mit Nullen auffüllen

Tabellenstruktur ändern

```
ALTER TABLE Tabellenname
  ADD [COLUMN] Spaltenname Typ [Optionen] [FIRST|AFTER Spaltenname]
| DROP [COLUMN] Spaltenname
| CHANGE [COLUMN] Spaltenname NeuerSpaltenname Typ [Optionen]
                                    [FIRST|AFTER Spaltenname]
| ADD [UNIQUE|FULLTEXT] INDEX [Indexname] [Indextyp]
                                    (Spaltenname[, ...])
| DROP INDEX Indexname
| ADD PRIMARY KEY (Spaltenname,...)
| DROP PRIMARY KEY
| RENAME [TO] NeuerTabellenname
```

Bedeutung der einzelnen Befehle

- `ADD [COLUM]` – Spalte hinzufügen
- `DROP [COLUMN]` – Spalte löschen
- `CHANGE [COLUMN]` – Spalte ändern/umbenennen/verschieben
- `ADD [...] INDEX` – Index hinzufügen
- `DROP INDEX` – Index entfernen
- `ADD PRIMARY KEY` – Primärschlüssel hinzufügen
- `DROP PRIMARY KEY` – Primärschlüssel entfernen
- `RENAME` – Tabelle umbenennen

Tabellen löschen

```
DROP TABLE Tabelle
```

Einfügeabfragen

```
INSERT INTO Tabelle [(Feld1, Feld2, ...)]
VALUES (Wert1, Wert2, ...)
    [, (Wert1, Wert2, ...), ...]
```

Auswahlabfragen

```
SELECT [DISTINCT[ROW]
   * | Spalte1[, Spalte2, ...] | Ausdruck[, Ausdruck, ...]
   [FROM [Datenbank.]Tabelle[, Tabelle, ...] | JOIN-Konstrukt]
   [WHERE Kriterium]
   [ORDER BY Spalte [ASC|DESC][, Spalte [ASC|DESC], ...]]
   [LIMIT Start,Anzahl]
```

JOIN-Konstrukt

```
Tabelle1 [INNER|LEFT|RIGHT|...] JOIN Tabelle2 ON JOIN-Kriterium
```

Wichtige Kriterien (für WHERE oder ON)

- Vergleich: `Spalte|Ausdruck=Spalte|Ausdruck`

  Mögliche Operatoren: `=`, `!=`, `<`, `>`, `<=`, `>=` und `<=>`
- Mustervergleich: `Spalte|Ausdruck LIKE "Muster"`

  Platzhalter im Muster: `_` (genau ein beliebiges Zeichen), `%` (beliebig viele beliebige Zeichen)
- Regulärer Ausdruck: `Spalte|Ausdruck REGEXP "RegulärerAusdruck"` (die RegEx-Syntax ist zu komplex für den Anhang, siehe Kapitel 6)
- Verknüpfung von Kriterien: `AND` (alle müssen zutreffen), `OR` (mindestens eins muss zutreffen), `XOR` (genau eins muss zutreffen), `NOT` (Kriterium darf nicht zutreffen)

Elemente für SQL-Ausdrücke

- Arithmetische Operationen: `+`, `-`, `*`, `/`, `%`
- Mathematische Funktionen, zum Beispiel: `SIN()`, `EXP()`, `ROUND()`, `RAND()`
- String-Funktionen, zum Beispiel: `CONCAT()`, `TRIM()`, `SUBSTRING()`
- Datums- und Uhrzeitfunktionen, zum Beispiel: `DATE_ADD()` und `DATE_FORMAT()`
- Aggregatfunktionen, zum Beispiel: `SUM()`, `MAX()`, `COUNT()` – für einzelne Datengruppen mittels `GROUP BY Spalte`

Datenänderungsabfragen

```
UPDATE [Datenbank.]Tabelle
SET Spalte1=Wert1[, Spalte2=Wert2 ...]
[WHERE Kriterium]
```

Löschabfragen

```
DELETE FROM [Datenbank.]Tabelle
[WHERE Kriterium]
```



Bei `UPDATE`- und `DELETE`-Abfragen müssen Sie darauf achten, dass Sie auf keinen Fall die `WHERE`-Klausel vergessen – andernfalls werden *alle* Datensätze der Tabelle geändert beziehungsweise gelöscht!

Informationen und Organisatorisches

Standarddatenbank festlegen:

```
USE Datenbank
```

Aktuelle Standarddatenbank anzeigen:

```
SELECT DATABASE()
```

Die Namen aller Datenbanken des MySQL-Servers anzeigen (optional nur diejenigen, die dem angegebenen `LIKE`-Suchmuster entsprechen):

```
SHOW DATABASES [LIKE "Muster"]
```

Die Namen aller Tabellen der Standarddatenbank anzeigen:

```
SHOW TABLES [LIKE "Muster"]
```

Die Struktur einer Tabelle anzeigen:

```
DESC[RIBE] Tabellenname
```

Eine `CREATE`-Anweisung anzeigen, die die angegebene Tabelle erstellen würde:

```
SHOW CREATE TABLE Tabellenname
```

MySQL-Funktionen in PHP

Tabelle A-1 zeigt eine Übersicht über die verschiedenen Funktionen beziehungsweise Methoden der beiden Schnittstellen *mysql* und *mysqli*.

Tabelle A-1: Die wichtigsten Elemente der Schnittstellen mysl und mysqli

Funktionalität	mysql	mysqli
Verbindungsaufbau	`$id = mysql_connect ($host,$user,$pass);`	`$conn = news mysqli ($host,$user,$pass,$db);`
Datenbankauswahl	`mysql_select_db ($db);`	`$conn->select_db ($db);`
Abfrage	`$query = mysql_query ($sql);`	`$query = $conn->query ($sql);`
Datensatz lesen (numerisches Array)	`$arr = mysql_fetch_ row ($query);`	`$arr = $query->fetch_row ();`
Datensatz lesen (benanntes Array)	`$arr = mysql_fetch_ array ($query);`	`$arr = $query->fetch_array ();`
Anzahl Ergebnisdatensätze (Auswahlabfrage)	`$lines = mysql_num_ rows($query);`	`$lines = $query->num_rows;`
Anzahl geänderter Zeilen (Änderungsabfragen)	`$lines = mysql_affected_ rows();`	`$lines = $conn->affected_rows`
Zuletzt eingefügte auto_ increment-ID	`$id = mysql_insert_id();`	`$id = $conn->insert_id;`

In Tabelle A-2 sehen Sie die entsprechenden Funktionen der Abstraktionsschnittstelle PHP Data Objects (PDO).

Tabelle A-2: Übersicht der wichtigsten PDO-Elemente

Funktionalität	PDO-Code
Verbindungsaufbau	`$conn = new PDO ("mysql:host=$host;dbname=$db", $user, $pass[, $option_array]);`
Datenbankauswahl	(nicht vorgesehen; bei Bedarf neue Verbindung herstellen)

Tabelle A-2: Übersicht der wichtigsten PDO-Elemente (Fortsetzung)

Funktionalität	PDO-Code
Abfrage	`$query = $conn->query ($sql);`
Datensatz lesen (numerisches Array)	`$arr = $query->fetch (PDO::FETCH_NUM);`
Datensatz lesen (benanntes Array)	`$arr = $query->fetch (PDO::FETCH_ASSOC);`
Anzahl Ergebnisdatensätze (Auswahlabfrage)	(nicht verfügbar; wenn Sie die Anzahl vorab benötigen, ist eine separate `SELECT COUNT(...)`-Abfrage erforderlich)
Anzahl geänderter Zeilen (Änderungsabfragen)	`$lines = $query->rowCount();`
Zuletzt eingefügte auto_increment-ID	`$id = $conn->lastInsertId();`
Verbindung schließen	`$conn = null;`

In diesem Anhang:

- Perl DBI
- Java JDBC
- Die MySQL-Schnittstelle in Ruby
- Ruby on Rails (Active Record)

ANHANG B
Sonstige APIs

Alle Sprache ist Bezeichnung der Gedanken.
Immanuel Kant

Neben PHP verfügen auch zahlreiche andere Programmiersprachen und Anwendungen über Schnittstellen zu MySQL-Datenbanken. Nachdem es in den Kapiteln ausschließlich um PHP ging, sollen in diesem Anhang exemplarisch zwei weitere Schnittstellen vorgestellt werden: Perl DBI und Java JDBC.

Perl DBI

Die Skriptsprache ist besonders bei Unix-Systemadministratoren, aber auch zur klassischen Webprogrammierung über die CGI-Schnittstelle beliebt. Sie enthält eine allgemeine Datenbankschnittstelle namens DBI (DataBase Interface). Für konkrete Datenbanken benötigt sie einen Treiber; derjenige für MySQL ist bereits ab Werk enthalten. Zu Beginn Ihres Perl-Skripts müssen Sie das Modul DBI einbinden:

```
use DBI;
```

Anschließend wird die DBI-Methode `connect()` aufgerufen, um ein Verbindungsobjekt zu erstellen. Sie benötigt drei Argumente gemäß dem folgenden Schema:

```
my $conn = DBI->connect
     ("dbi:mysql:Datenbank;Host", $username, $passwort);
```

Hier ein Beispiel, das mit den aus Kapitel 8 bekannten Anmeldedaten eine Verbindung zur Datenbank *reisebuero* herstellt:

```
my $conn = DBI->connect
     ("dbi:mysql:reisebuero;localhost", "rbuser", "R3153n");
```

Abfragen werden zunächst mithilfe der Methode `prepare()` vorbereitet und anschließend mit `execute()` ausgeführt. Die Syntax sieht wie folgt aus:

```
my $query = $conn->prepare ($querytext);
$query->execute();
```

Das folgende Beispiel wählt alle Felder aller Hotels aus:

```
my $query = $conn->prepare ("SELECT * FROM rb_hotels");
$query->execute();
```

Mithilfe der Methode `$query->fetchrow_array()` können Sie die Daten dann in einer `while`-Schleife auslesen. Zu guter Letzt können Sie die Methode `disconnect()` aufrufen, um die Verbindung zu beenden.

Hier ein kleines, vollständiges Beispiel, das die Hotels sowie die Städte, in denen sie sich befinden, sortiert auf der Konsole ausgibt:

```
#!/usr/bin/perl -w
use strict;
use DBI;
my $conn = DBI->connect ('dbi:mysql:reisebuero;localhost', 'rbuser', 'R3153n');
my $querytext = << "ENDSQL";

SELECT ht_name, st_name FROM
rb_hotels INNER JOIN rb_staedte ON ht_stadt=st_nr
ORDER BY st_name ASC, ht_name ASC

ENDSQL

my $query = $conn->prepare ($querytext);
$query->execute();
while (my ($hotel, $stadt) = $query->fetchrow_array()) {
   printf ("%s in %s\n", $hotel, $stadt);
}

$conn->disconnect();
```

Java JDBC

Auch die Programmiersprache Java verfügt über eine eigene Bibliothek mit Datenbankschnittstellen namens JDBC. Für MySQL benötigen Sie den Treiber *MySQL/ConnectorJ*, der auf der Website *http://www.mysql.com* unter *Connectors* heruntergeladen werden kann, zurzeit in Version 3.1 (auf der beiliegenden CD-ROM ist er im Verzeichnis *tools* zu finden).

Zu Beginn einer Java-Anwendung, die JDBC verwenden soll, müssen Sie die JDBC-Klassen importieren:

```
import java.sql.*;
```

Innerhalb des Programms wird als Nächstes der MySQL-Treiber geladen; der Fehler, dass er nicht vorhanden ist, muss per `try/catch` abgefangen werden:

```
try {
   Class.forName ("com.mysql.jdbc.Driver").newInstance();
}
catch (ClassNotFoundException e) {
   System.out.println ("MySQL/ConnectorJ nicht gefunden");
}
```

Beachten Sie, dass alle datenbankbezogenen Operationen ebenfalls von einem try/catch-Block umschlossen werden müssen; in diesem Fall muss eine SQLException, also ein Datenbankfehler, abgefangen werden.

Anschließend wird mithilfe der Klasse java.sql.DriverManager eine Datenbankverbindung hergestellt. Das sieht schematisch folgendermaßen aus:

```
Connection conn = DriverManager.getConnection
     ("jdbc:mysql://Host/Datenbank", "Benutzername", "Passwort");
```

Hier ein Beispiel für die Reisebüro-Datenbank:

```
Connection conn = DriverManager.getConnection
     ("jdbc:mysql://localhost/reisebuero", "rbuser", "R3153n");
```

Für Abfragen wird ein Statement-Objekt benötigt:

```
Statement st = conn.createStatement();
```

Abfragen können Sie anschließend über die Methoden executeQuery() beziehungsweise execute() erstellen. executeQuery() liefert ein ResultSet zurück und ist damit für Auswahlabfragen mit Ergebnisdatensätzen geeignet; execute() wird dagegen für alle anderen Abfragen verwendet. Beispiele:

```
// Alle Daten aller Hotels auslesen
ResultSet rs = st.executeQuery ("SELECT * FROM rb_hotels");
// Doppelzimmer in Kölner Hotels um 10 Euro billiger machen
st.execute
  ("UPDATE rb_hotels SET ht_dzpreis=ht_dzpreis-10 WHERE ht_stadt=1");
```

Das ResultSet enthält die Ergebnisdatensätze der SELECT-Abfrage und kann mithilfe der Methode next() in einer while-Schleife durchgeblättert werden. Eine Reihe von get*()-Methoden liefert die einzelnen Felder eines Datensatzes: Da Java eine typisierte Sprache ist, werden die verschiedenen SQL-Datentypen auf unterschiedliche Java-Typen abgebildet, zum Beispiel getInt() für Ganzzahlen oder getString() für Zeichenketten.

Hier ein vollständiges Beispiel, das die Namen, Kürzel und Länder aller Fluggesellschaften auf der Konsole ausgibt:

```
import java.sql.*;

public class AirlineTest {
   public static void main (String[] args) {
      // MySQL/ConnectorJ laden
      try {
         Class.forName ("com.mysql.jdbc.Driver");
      }
      catch (ClassNotFoundException e) {
         System.out.println ("MySQL/ConnectorJ nicht gefunden");
      }
            // Ergebnisvariablen
      String airport;
      String kuerzel;
      String land;
```

```
        try {
           // Verbindung herstellen
           Connection conn = DriverManager.getConnection
              ("jdbc:mysql://localhost/reisebuero",
               "rbuser", "R3153n");

           // Statement für Abfragen
           Statement st = conn.createStatement();

           // Abfragetext
           String qtext = "SELECT ai_name, ai_kuerzel, la_name
             FROM rb_airlines INNER JOIN rb_laender ON ai_land=la_nr";

           // Abfrage senden
           ResultSet rs = st.executeQuery (qtext);

           // Ergebnis in einer Schleife ausgeben
           while (rs.next()) {
              airport = rs.getString ("ai_name");
              kuerzel = rs.getString ("ai_kuerzel");
              land = rs.getString ("la_name");
              System.out.println
                   (airport + " (" + kuerzel + ")" + ", " + land);
           }
           // Statement und Verbindung schließen
           st.close();
           conn.close();
        }
        catch (SQLException e) {
           System.out.println ("Datenbankfehler");
        }
     }
  }
```

Die MySQL-Schnittstelle in Ruby

Die objektorientierte Skriptsprache Ruby (Informationen und Download unter *http://www.ruby-lang.org*) besitzt eine optionale MySQL-Schnittstelle; sie ähnelt stark der PHP-Schnittstelle mysqli. Das Folgende ist eine (hier an das Reisebüro-Beispiel angepasste) Kurzfassung des entsprechenden Abschnitts aus meinem Buch *Praxiswissen Ruby*, das 2007 ebenfalls in der vorliegenden O'Reilly-Buchreihe erschienen ist und noch mehr Informationen über datenbankbasierte Ruby-Anwendungen am Beispiel MySQL enthält.

Für Windows kann die MySQL-Schnittstelle wie folgt über den Ruby-Erweiterungsmanager rubygems installiert werden:

```
> gem update
> gem install mysql --include-dependencies
```

Um die Erweiterung unter Unix zu installieren, müssen Sie sie dagegen aus dem Quellcode-Paket installieren. Laden Sie sie dazu von *http://tmtm.org/downloads/mysql/ruby/* herunter; die aktuelle Version ist derzeit 2.7. Entpacken Sie das Archiv wie folgt:

```
# tar xzvf mysql-ruby-2.7.tar.gz
```

Wechseln Sie danach in das neue Unterverzeichnis:

```
# cd mysql-ruby-2.7
```

Nun wird das Skript *extconf.rb* unter Angabe Ihres MySQL-Verzeichnisses aufgerufen, zum Beispiel:

```
# ruby extconf.rb /usr/local/mysql
```

Zum Schluss werden `make` und `make install` aufgerufen:

```
# make
# make install
```

Der MySQL-Zugriff über die Bibliothek `mysql` erfordert zunächst die passenden `require`-Anweisungen:

```
require "rubygems"
require "mysql"
```

Anschließend können Sie den Konstruktor der Klasse `Mysql` aufrufen, um eine Verbindung zum Datenbankserver herzustellen. Die notwendigen Parameter sind Host, Benutzername und Passwort. Das folgende Beispiel stellt eine Verbindung mit den Rechten des Benutzers `rbuser` her:

```
conn = Mysql.new("localhost", "rbuser", "R3153n")
```

Anschließend sollten Sie wie üblich die Standarddatenbank wählen, mit der gearbeitet werden soll. Dies geschieht mithilfe der Methode `select_db`. Die Datenbank *reisebuero* wird beispielsweise wie folgt ausgewählt:

```
conn.select_db("reisebuero")
```

Nun lässt sich die Verbindung ganz einfach für Datenbankabfragen verwenden. Auswahlabfragen mit `SELECT`, die Datensätze zurückliefern, sollten Sie einer Variablen zuweisen. Diese hat den Datentyp `Mysql::Result` und enthält verschiedene Methoden zum Auslesen des Ergebnisses. Das folgende Beispiel ermittelt die Kombination aus Hotels und Städten:

```
result = conn.query(
  "SELECT ht_name, st_name
  FROM rb_hotels INNER JOIN rb_staedte
  ON ht_stadt=st_nr"
)
```

Die erste Methode von `result`, die Sie aufrufen sollten, ist `num_rows`. Sie liefert die Anzahl der Ergebnisdatensätze zurück. Wenn sie 0 ist, entsprach kein Datensatz Ihrer Abfrage, zum Beispiel:

```
if result.num_rows > 0
  # Ergebnis ausgeben
else
  puts "Keine Datens&auml;tze gefunden."
end
```

Zum Auslesen der Ergebnisse kommen vor allem zwei Methoden in Frage:

- `fetch_row` liest einen Datensatz als nummeriertes Array aus. Die Reihenfolge der Felder innerhalb einer Ergebniszeile entspricht den Angaben in Ihrer Abfrage; bei * (alle Felder) wird die Reihenfolge der Tabelle selbst gewählt.
- `fetch_hash` liest ebenfalls genau einen Datensatz aus. Der einzige Unterschied besteht darin, dass Sie einen Hash zurückerhalten, in dem die Spaltennamen aus den Datenbanktabellen die Schlüssel bilden.

Beide Methoden lassen sich idealerweise innerhalb der Bedingung einer `while`-Schleife platzieren, um alle Datensätze nacheinander auszulesen.

Hier ein Beispiel für den Einsatz von `fetch_row`:

```
while line = result.fetch_row
  printf "%s in %s\n", line[0], line[1]
end
```

Das müsste folgende Ausgabe liefern:

```
Bergerhof in Köln
Hotel Colonia in Köln
Hotel de la Gare in Paris
Hotel au Jardin in Paris
Otel Bahar in Istanbul
...
```

Mit `fetch_hash` funktioniert die Abfrage im Prinzip genauso, bis auf den besser lesbaren Zugriff auf die Felder:

```
while line = result.fetch_hash
  printf "%s in %s\n",
         line['ht_name'], line['st_name']
end
```

Bei Änderungsabfragen, die keine Datensätze zurückliefern, brauchen Sie keine Ergebnisvariable. Stattdessen können Sie nach Durchführung der Abfrage die Eigenschaft `affected_rows` Ihres Verbindungsobjekts auslesen. Sie gibt an, wie viele Datensätze geändert wurden. Das folgende Beispiel fügt einen neuen Kunden zur Kundentabelle hinzu und gibt danach an, ob es funktioniert hat:

```
conn.query
  ("INSERT INTO rb_kunden (kd_mail) VALUES ('new@test.com')")
if conn.affected_rows > 0
  puts "Kunde erfolgreich hinzugef&uuml;gt."
else
  puts "Kunde konnte nicht hinzugef&uuml;gt werden."
end
```

Alle MySQL-Methoden lösen Exceptions vom Typ `Mysql::Error` aus. Wenn Sie diese abfangen, können Sie nützliche Informationen über den genauen Fehler erhalten. Das folgende Beispiel versucht, Daten aus einer nicht existierenden Tabelle auszulesen:

```
begin
  result = conn.query("SELECT * FROM rb_airline")
rescue Mysql::Error => e
  puts "Fehlernummer:  #{e.errno}"     # Fehlercode
  puts "Fehlermeldung: #{e.error}"     # Meldungstext
  puts "SQL-Zustand:   #{e.sqlstate}"  # Fehler nach SQL-Standard
end
```

Sie erhalten folgende Ausgabe:

```
Fehlernummer:  1146
Fehlermeldung: Table 'reisebuero.rb_airline' doesn't exist
SQL-Zustand:   42S02
```

Sobald Sie die MySQL-Verbindung nicht mehr benötigen, sollten Sie sie schließen:

```
conn.close
```

Ruby on Rails (Active Record)

Ruby on Rails (kurz *Rails*), einer der wichtigsten Gründe für den derzeitigen Popularitätsschub von Ruby, ist ein modernes *Web-Framework*. Es ermöglicht die Erstellung von Webanwendungen gemäß dem sogenannten MVC-Muster (Model, View, Controller). Das bedeutet, dass Datenstruktur, Programmierlogik und Benutzeroberfläche sauber voneinander getrennt werden und einzeln ausgetauscht werden können. Das Model wird dabei in der Regel in einer Datenbank gespeichert, wobei MySQL Standard ist. Beachten Sie, dass dazu die im vorigen Abschnitt gezeigte MySQL-Erweiterung in Ruby installiert sein muss.

Die Datenbankverbindung in Ruby on Rails wird über eine Komponente namens Active Record hergestellt. Es handelt sich um einen sogenannten *objektrelationalen Mapper* (ORM), der die relationalen Datenbanktabellen automatisch mit einer Objekthierarchie verknüpft.

Die Funktionsweise von Active Record können Sie nur nachvollziehen, wenn Sie sich eine komplette Rails-Anwendung anschauen. Das nachfolgende (stark gekürzte) Komplettbeispiel stammt ebenfalls aus meinem Buch *Praxiswissen Ruby*; es handelt sich um eine kleine Anwendung, die Rockbands und ihre Alben zeigt.[1]

1 Wenn Sie noch mehr über Ruby on Rails wissen möchten, schauen Sie sich das Buch *Praxiswissen Ruby on Rails* von Denny Carl an (O'Reilly Verlag).

Zunächst müssen Sie Rails selbst installieren. Wenn Sie bereits Ruby haben, gelingt dies leicht mithilfe der folgenden Kommandozeilenanweisungen:

```
> gem update
> gem install mysql --include-dependencies
```

Der erste Schritt besteht darin, das Skelett der Anwendung zu erzeugen. Erstellen Sie ein Verzeichnis für Ihre Rails-Anwendungen und geben Sie darin Folgendes ein:

```
> rails rock_n_roll
```

Dadurch wird automatisch eine Verzeichnisstruktur erzeugt, deren Elemente angezeigt werden. Die wichtigsten Verzeichnisse haben folgende Bedeutung:

- *app* enthält die Komponenten der eigentlichen Anwendung, unterteilt in die drei oben genannten MVC-Aspekte *models*, *views* und *controllers* sowie das zusätzliche Unterverzeichnis *helpers* für gemeinsame Hilfsklassen.
- *config* enthält Konfigurationsdateien. Rails verwendet zwar ein Prinzip namens »Convention over Configuration«, aber ganz ohne Konfigurationsdateien geht es nun einmal nicht. Meist brauchen Sie nur die Datei *database.yml* zu editieren, die die Namen und Zugangsdaten der verwendeten Datenbanken enthält.
- *script* enthält einige vorgefertigte Hilfsskripten, die nicht innerhalb Ihrer Rails-Anwendung ausgeführt werden, sondern Ihnen bei der Entwicklung helfen. Das Skript *generate* erzeugt beispielsweise die Grundgerüste Ihrer MVC-Komponenten, und *server* ist der *WEBrick*-Server – ein kleiner Webserver, der die Anwendung zu Testzwecken ausführt.
- *public* enthält eine CGI-Konfiguration. Mit ihrer Hilfe kann die Rails-Anwendung praktisch von jedem CGI-fähigen Webserver bereitgestellt werden. Weiter unten wird natürlich die entsprechende Konfiguration von Apache erläutert.
- *test* stellt eine Umgebung für sogenannte *Unit-Tests* bereit, ein beliebtes Mittel zum automatisierten Testen objektorientierter Programme.

Die zweite Aufgabe besteht darin, das Model zu erstellen. Sie umfasst vier Einzelschritte:

- Erzeugen der Datenbanken und Tabellen.
- Eintragen der Datenbanknamen und -zugriffsparameter in die Konfigurationsdatei *config/database.yml*.
- Automatisches Generieren der zugehörigen Model-Klassen.
- Anpassen der erzeugten Klassen durch Hinzufügen der Tabellenrelationen.

Starten Sie für den ersten Schritt den Konsolenclient mysql:

```
> mysql -u root -p
[Passwort]
```

Eine Rails-Anwendung benötigt bis zu drei Datenbanken, die standardmäßig mit *Anwendungsname_development*, *Anwendungsname_test* und *Anwendungsname_production* bezeichnet werden. Sie sind für die Entwicklung, die Unit-Tests beziehungsweise die Produktion (die tatsächliche Veröffentlichung der Site) zuständig. Zu Beginn benötigen Sie nur die Entwicklungs- und gegebenenfalls die Testdatenbank. Es schadet aber nichts, gleich alle drei zu erstellen. Da die neue Anwendung *rock_n_roll* heißt, können Sie dazu die folgenden drei Zeilen eingeben:

```
mysql> CREATE DATABASE rock_n_roll_development;
mysql> CREATE DATABASE rock_n_roll_test;
mysql> CREATE DATABASE rock_n_roll_production;
```

Als Nächstes sollten Sie einen separaten MySQL-User erstellen, der Zugriff auf diese Datenbanken hat. Benutzername und Passwort dieses Benutzerkontos müssen Sie in die Konfiguration der Anwendung schreiben. In der Praxis verwenden Sie aus Sicherheitsgründen besser einen separaten User für die Produktionsdatenbank, aber zum Ausprobieren soll erst einmal ein gemeinsames Konto genügen. Die folgenden Zeilen legen einen User namens *rock_user* mit dem Passwort *rockonrails* an, der vollen Zugriff auf alle drei Datenbanken besitzt:

```
mysql> CREATE USER rock_user@localhost IDENTIFIED BY "rockonrails";
mysql> GRANT ALL ON rock_n_roll_development.* TO rock_user@localhost;
mysql> GRANT ALL ON rock_n_roll_test.* TO rock_user@localhost;
mysql> GRANT ALL ON rock_n_roll_production.* TO rock_user@localhost;
```

Anschließend sollten Sie die MySQL-Benutzerdaten aktualisieren, damit diese Änderungen wirksam werden:

```
mysql> FLUSH PRIVILEGES;
```

Jetzt können die Tabellen erstellt werden, die das Model der Anwendung bilden sollen. Hier sollten konsequent englische Namen zum Einsatz kommen, weil Rails bei Beziehungen zwischen Tabellen automatisch den Plural beziehungsweise Singular englischer Standardwörter bildet (»one band has many albums« oder formaler: `band.has_many :albums`) – bis hin zu eingebauten Ausnahmen wie *person* <=> *people*. Wenn Sie deutsche (oder sehr seltene englische) Namen verwenden möchten, müssen Sie die Pluralregeln einzeln in *config/environment.rb* angeben.

Zunächst brauchen Sie nur in der Entwicklungsdatenbank Tabellen. Wählen Sie diese als Standarddatenbank aus:

```
mysql> USE rock_n_roll_development
```

Die beiden Tabellen sollen *bands* und *albums* heißen. Eine Band wird dabei durch ihren Namen und ihr Herkunftsland charakterisiert, ein Album durch Titel, Erscheinungsjahr und Band. Letzteres ist eine Relation mit einem Datensatz in *bands*. Da eine Band beliebig viele Alben produzieren kann, handelt es sich um eine 1:n-Relation. Die Rails-Konventionen gehen davon aus, dass jede Tabelle einen Primärschlüssel namens `id` besitzt und dass ein Feld mit Bezug auf eine andere Tabelle einen Namen nach dem Schema `andereTabelle_id` erhält.

Geben Sie also folgenden SQL-Code ein, um die Tabellen anzulegen:

```
mysql> CREATE TABLE bands (
    ->   id INT AUTO_INCREMENT PRIMARY KEY,
    ->   name VARCHAR(40),
    ->   country VARCHAR(40)
    -> );

mysql> CREATE TABLE albums (
    ->   id INT AUTO_INCREMENT PRIMARY KEY,
    ->   title VARCHAR(40),
    ->   release_year DATE,
    ->   band_id INT
    -> );
```

Verlassen Sie den MySQL-Client nach diesen Eingaben oder geben Sie vorher ein paar Testdaten ein, wenn Sie möchten. Schon bald werden Sie allerdings auch ein praktisches Web-Interface zur Datenpflege erhalten.

Nun ist es Zeit, die Datei *config/database.yml* in einem Texteditor zu öffnen und die Datenbankverbindungsparameter einzugeben. Die Datei ist im YAML-Format geschrieben. Es handelt sich dabei um eine sehr einfache Sprache zur Darstellung verschiedener Datenstrukturen in Textform. Das Format ist knapper und weniger komplex als etwa XML.

Die `development`-Sektion der automatisch erstellten Version der Konfigurationsdatei sieht so aus (die Kommentare wurden entfernt):

```
development:
  adapter: mysql
  database: rock_n_roll_development
  username: root
  password:
  host: localhost
```

Wie Sie sehen, ist ein großer Teil der Arbeit bereits getan. Sogar der Datenbankadapter `mysql` ist bereits eingestellt, da er die Standardvorgabe für Rails ist. Ersetzen Sie den `username` `root` durch `rock_user` und fügen Sie das Passwort `rockonrails` hinzu:

```
development:
  adapter: mysql
  database: rock_n_roll_development
  username: rock_user
  password: rockonrails
  host: localhost
```

Nachdem Sie die geänderte Datei gespeichert haben, können die Model-Klassen erzeugt werden. Geben Sie dazu folgende Kommandos ein:

```
> ruby script/generate model Band
> ruby script/generate model Album
```

Die erstellten Klassen sind sogenannte *Stubs* (englisch für Stummel oder Stumpf). Sie enthalten also die Basisfunktionalität ohne irgendwelche Extras. Das Wichtigste, das Sie hinzufügen müssen, sind die Relationen zwischen den Daten.

Öffnen Sie also zunächst die Datei *app/models/bands.rb*. Sie besteht nur aus den folgenden beiden Zeilen:

```
class Bands < ActiveRecord::Base
end
```

Die Klasse `Bands`, die mit der gleichnamigen (kleingeschriebenen) Datenbanktabelle korrespondiert, wird also von `ActiveRecord::Base` abgeleitet, wo alles Wesentliche für die Umsetzung der Datensätze in Ruby-Instanzen bereits existiert. Ergänzen Sie die Klassendefinition wie folgt, um klarzustellen, dass eine Band mehrere Alben veröffentlichen kann:

```
class Bands < ActiveRecord::Base
  has_many :albums
end
```

Auch *app/models/albums.rb* müssen Sie entsprechend erweitern. Hier lautet die passende Beziehung `belongs_to`, weil ein Album stets einer bestimmten Band zugeordnet ist (der Sonderfall Sampler wird hier nicht betrachtet). Die ursprüngliche Fassung

```
class Albums < ActiveRecord::Base
end
```

wird somit zu

```
class Albums < ActiveRecord::Base
  belongs_to :band
end
```

Beachten Sie den Singular bei `belongs_to :band`.

Was an dieser Stelle nicht behandelt wird, ist die Möglichkeit, Validierungsanforderungen hinzuzufügen. Diese überprüfen beim Einfügen oder Ändern von Daten automatisch, ob sie beispielsweise existieren oder ein bestimmtes Format besitzen.

Eine besonders beeindruckende Fähigkeit von Ruby on Rails ist das *Scaffolding* (englisch für Gerüstbau). Mit einer einzigen Kommandozeilenanweisung erstellt es Ihnen eine Webschnittstelle zum Erstellen, Lesen, Ändern und Löschen von Datensätzen einer Tabelle (kurz *CRUD* für Create, Read, Update, Delete).

Im vorliegenden Beispiel können Sie das Scaffolding nur für die Tabelle *bands* benutzen, weil das Verfahren noch nicht »intelligent« genug ist, Tabellenrelationen automatisch zu berücksichtigen. Geben Sie also Folgendes ein:

```
> ruby script/generate scaffold band
```

Starten Sie nun den WEBrick-Server der Anwendung und geben Sie die URL *http://localhost:3000/bands* in Ihren Browser ein. Zunächst sehen Sie eine leere Liste. Kli-

cken Sie auf *New band*, geben Sie eine Band Ihrer Wahl sowie deren Herkunftsland ein, und klicken Sie auf dann auf *Create*. Wiederholen Sie das ein paarmal, bis Sie eine Liste erhalten.

Im Verzeichnis *script* Ihrer Rails-Anwendung finden Sie noch zwei weitere praktische Helfer. *scripts/runner* führt einzelne Ruby-Anweisungen in der Rails-Umgebung aus. *scripts/console* ruft dagegen die interaktive Ruby-Shell `irb` auf, in der Sie beliebig viele Ruby-Anweisungen direkt ausführen können, und stellt darin sämtliche Klassen der Ruby on Rails-Anwendung zur Verfügung.

Probieren Sie zuerst *scripts/runner* aus. Das folgende Beispiel liest den Namen und das Land der ersten Band aus der Tabelle *bands*:

```
> ruby script/runner "b1 = Band.find(1); puts b1.name +
' (' + b1.country + ')'"
Metallica (USA)
```

Die Klasse `Band` wurde automatisch aus der Tabelle *bands* generiert. Ihre Klassenmethode `find` sucht nach einer konkreten Instanz (oder einem Datensatz) mit der angegebenen ID, die in der Datenbanktabelle wie bereits erwähnt `id` heißen muss. Die verschiedenen Felder schließlich stehen einfach als Methoden der gefundenen Instanz zur Verfügung.

Starten Sie für den nächsten Versuch *scripts/console*:

```
> ruby script/console
Loading development environment.
```

Geben Sie folgenden Code ein, um eine Liste aller Bands mit ihren IDs, Namen und Ländern zu erhalten:

```
>> Band.find(:all).each { |b|
?>   puts "#{b.id}. #{b.name} (#{b.country})"
>> }
1. Metallica (USA)
2. Iron Maiden (United Kingdom)
3. Die Ärzte (Germany)
4. Led Zeppelin (United Kingdom)
5. Die Toten Hosen (Germany)
6. Extreme (USA)
```

Das Symbol `:all` kann statt einer konkreten ID verwendet werden, um alle Datensätze auszulesen.

Zum Schluss sollten Sie einige Alben anlegen. Auch für die Neuerstellung von Datensätzen gibt es eine einfache Vorgehensweise:

```
Klasse.create(:attr1 => Wert, :attr2 => Wert, ...)
```

Das Einfügen von Datensätzen ist mit anderen Worten nur noch ein Ruby-Methodenaufruf. Mit der nummerierten Bandliste ist nun auch bekannt, welche Band zu welchem Album eingegeben werden muss. Legen Sie also beispielsweiese damit los:

```
>> Album.create(:title => "Die Bestie in Menschengestalt",
?> :release_year => "1993", :band_id => 3)
```

Wenn Sie einen Fehler machen, können Sie zum Beispiel die Methode `update_attribute` eines Datensatzobjekts aufrufen, um ihn zu beheben. Die benötigten Attribute sind `Feldname` und `neuer Wert`. Angenommen, Sie legen das Metallica-Album »Metallica« (das sogenannte »Black Album«) an und geben als Jahr 1990 statt 1991 ein:

```
>> Album.create(:title => "Metallica",
?> :release_year => "1990", :band_id => 1)
```

Um die Daten eines konkreten Albums zu ändern, müssen Sie das entsprechende Album zunächst einmal finden. Statt einer bestimmten ID können Sie bei `find` auch die Kombination `:all` und `:conditions => "Bedingungen ..."` angeben. Die Bedingungen sind praktisch derjenige Teil einer SQL-Abfrage, den Sie in einem `SELECT` hinter das `WHERE` schreiben würden, zum Beispiel:

```
>> Album.find(:all, :conditions => "title like 'Met%'").each { |a|
?>   puts a.title
>> }
Metallica
```

Mithilfe dieser Information können Sie das Jahr korrigieren:

```
>> Album.find(:all, :conditions => "title like 'Met%'").each { |a|
?>   a.update_attribute(:release_year, "1991")
>> }
=> [#<Album:0x381cccc @attributes={"title"=>"Metallica",
   "release_year"=>"1991", "id"=>"5", "band_id"=>"1"}>]
```

Geben Sie einige weitere Alben Ihrer Wahl ein und verwenden Sie zum Schluss diesen Code, um sich eine Liste der Alben mit dem Namen ihrer jeweiligen Band anzeigen zu lassen:

```
>> Album.find(:all).each { |a|
?>   b = Band.find(a.band_id)
>>   puts "'#{a.title}' von #{b.name} (#{a.release_year})"
>> }
'Master of Puppets' von Metallica (1986)
'Ride The Lightning' von Metallica (1984)
'The Number Of The Beast' von Iron Maiden (1982)
'Die Bestie in Menschengestalt' von Die Ärzte (1993)
'Metallica' von Metallica (1991)
```

Als Nächstes werden Controller und View der Rock-and-Roll-Anwendung erstellt. Die Bands und ihre Alben sollen als verschachtelte Liste angezeigt werden.

Erstellen Sie als Erstes das Grundgerüst des Album-Controllers:

```
> ruby script/generate controller Album
```

Zuerst soll die Ansicht erstellt werden:

```
> ruby script/generate controller Album list
```

Sie werden gefragt, ob die bestehende Datei überschrieben werden soll. Die Antwortmöglichkeiten sind `y` (ja), `n` (nein), `a` (alle weiteren überschreiben) oder `q` (Erzeugung abbrechen). Im vorliegenden Fall sollten Sie `y` wählen.

Erstellen Sie nun die Controller-Methode `list` für die Listenansicht. Wenn Sie die Datei *app/controllers/album_controller.rb* öffnen, finden Sie dort die Klassendefinition mit dem Methoden-Stub `list`:

```
class AlbumController < ApplicationController

  def list
  end

end
```

Ergänzen Sie `list` zu folgendem Gesamtcode – die Active Record-Methodenaufrufe sollten Ihnen nach den obigen Erläuterungen bekannt sein:

```
  def list
    # Listeninhalt privat erstellen
    bandlist = "<ul>"
    # Über alle Bands iterieren
    Band.find(:all).each { |b|
      bandlist += "<li><b>#{b.name} (#{b.country})</b></li>"
      # Über alle Alben der Band iterieren
      bandlist += "<ul>"
      Album.find(:all, :conditions => "band_id=#{b.id}").each { |a|
        bandlist += "<li>#{a.title} (#{a.release_year})</li>"
      }
      bandlist += "</ul>"
    }
    bandlist += "</ul>"

    # Liste als Instanzvariable veröffentlichen
    @bandlist = bandlist
  end
```

Nun müssen Sie nur noch die entsprechende View – *app/views/list.rhtml* – ergänzen. Sie hat momentan folgenden Inhalt:

```
<h1>Album#list</h1>
<p>Find me in app/views/album/list.rhtml</p>
```

Erstellen Sie stattdessen den folgenden HTML-Code mit eingebettetem Ruby (sogenanntem eRuby):

```
<html>
<head>
<title>Rock'n'Rails -- Bands und ihre Alben</title>
</head>
<body>
<h1>Alle Bands und ihre Alben</h1>
<p><%= @bandlist %></p>
</body>
</html>
```

Wie Sie sehen, brauchen Sie nicht mehr eRuby-Code als die Ausgabe der Instanzvariablen `@bandlist`. Starten Sie nun den WEBrick-Server:

```
> ruby rock_n_roll/script/server
```

Geben Sie anschließend die URL *http://localhost:3000/album/list* in einen Browser ein.

In diesem Anhang:

- MySQL Administrator
- MySQL Query Browser

ANHANG C
Weitere Clients

It is a mistake to think you can solve any major problems just with potatoes.
Douglas Adams

In diesem Buch wurden zwei MySQL-Clients ausführlicher vorgestellt: der Kommandozeilenclient mysql und die webbasierte Lösung phpMyAdmin. Neben diesen beiden gibt es zahlreiche weitere Programme zum Bearbeiten von MySQL-Datenbanken. Zwei von ihnen – MySQL Administrator und MySQL Query Browser – stammen von den MySQL-Entwicklern selbst. Diese werden hier kurz vorgestellt.

MySQL Administrator

Das Tool *MySQL Administrator* dient der Steuerung und Administration der MySQL-Serversoftware selbst. Unter anderem bietet es eine grafische Konfigurationsmöglichkeit einiger Tools, die in Kapitel 9 behandelt wurden.

Beim ersten Start von MySQL Administrator müssen Sie sich wie üblich am MySQL-Server anmelden – für dieses Programm sinnvollerweise als *root*. Die Login-Daten können auf Wunsch gespeichert werden.

In der linken Spalte des Fensters von MySQL Administrator stehen verschiedene Hauptkategorien zur Auswahl, deren Konfigurationsmöglichkeiten dann jeweils rechts im Hauptbereich angezeigt werden. Die meisten Kategorien bestehen dabei aus mehreren Registerkarten. Hier eine kurze Übersicht über die Kategorien:

- *Server Information* liefert die wichtigsten Statusinformationen auf einen Blick. Hier erfahren Sie, ob der Server grundsätzlich arbeitet, und erhalten einen Überblick über seine Netzwerk- und Clientanbindung.
- *Service Control* (nur Windows) ermöglicht die Steuerung des MySQL-Systemdiensts. Hier können Sie also festlegen, ob und mit welchen Optionen der MySQL-Server beim Booten automatisch gestartet werden soll. Zudem können Sie ihn hier direkt starten und stoppen.

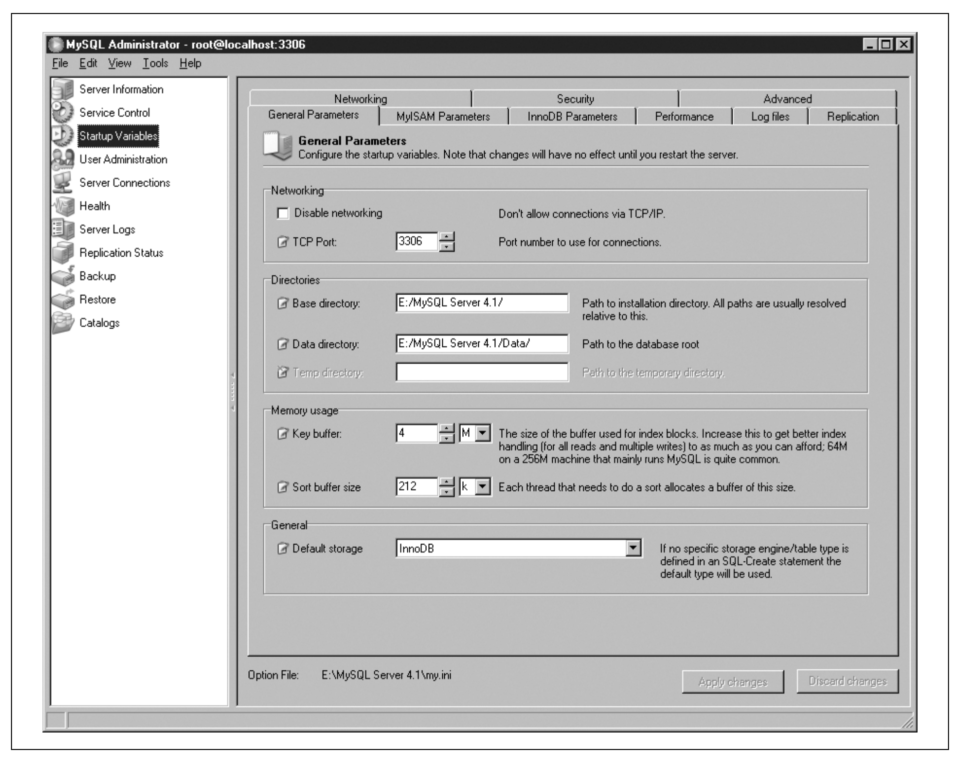


Abbildung C-1: MySQL Administrator mit der Seite »General Parameters« in der Kategorie »Startup Variables«

- *Startup Variables* ermöglicht die Konfiguration zahlreicher Parameter des Servers. Unter *General Parameters* (siehe Abbildung C-1) werden beispielsweise der TCP-Port, die Arbeitsverzeichnisse und der Standardtabellentyp eingestellt. Weitere Registerkarten betreffen etwa Netzwerk-, Sicherheits-, MyISAM- oder InnoDB-Einstellungen.
- *User Administration* dient der Verwaltung von Benutzerkonten (Registerkarte *User Information*) und Berechtigungen (*Schema Privileges*).
- *Server Connections* zeigt eine tabellarische Übersicht über die laufenden Clientverbindungen des MySQL-Servers an.
- *Health* gibt diverse Auslastungsdiagramme für Clientverbindungen und Speichernutzung aus. Auf weiteren Registerkarten werden die Werte der wichtigsten Status- und Systemvariablen in einer übersichtlichen Baumstruktur angezeigt.
- *Server Logs* ermöglicht eine grafisch unterstützte Untersuchung der MySQL-Logdateien.
- *Replication Status* zeigt sämtliche Replikationseinstellungen in einer tabellarischen Übersicht an.

- *Backup* dient der Datensicherung Ihrer MySQL-Datenbanken. Auf der Registerkarte *Backup Project* können Sie zunächst beliebige Sicherungseinstellungen zu Projekten zusammenfassen und gegebenenfalls manuell ausführen. *Advanced Options* dient der Auswahl der genauen Backup-Methode. Unter *Schedule* kann schließlich ein Zeitplan für die automatische Ausführung der Backup-Projekte festgelegt werden.
- *Restore* ermöglicht die Wiederherstellung eines Backups im Fall eines Fehlers oder Datenverlusts.
- *Catalogs* zeigt zunächst sämtliche Datenbanken an. Sobald Sie eine auswählen, werden ihre Tabellen gezeigt. Anschließend können Sie die Datenstruktur und die Optionen der jeweiligen Tabelle modifizieren.

MySQL Query Browser

Der MySQL Query Browser stammt ebenfalls von den MySQL-Entwicklern. Seine Hauptaufgabe ist das komfortable Erstellen und Ausführen beliebiger SQL-Abfragen; hier geht es also um die Verwaltung der Datenbanken selbst.

Auch der Query Browser verlangt beim Start eine Anmeldung. Anschließend wird das Hauptfenster des Programms angezeigt. Das Feld im oberen Bereich dient der Eingabe einer SQL-Abfrage, die Sie mithilfe der Schaltfläche *Execute* ausführen können. Rechts werden die *Schemata*, das heißt eine Baumansicht der Datenbanken, Tabellen und Spalten, angezeigt. Ein Doppelklick auf eine Datenbank wählt diese als Standard aus, so dass Sie sich in Abfragen ohne Angabe des Datenbanknamens auf ihre Tabellen beziehen können. Wenn Sie dagegen auf eine Tabelle doppelklicken, wird in das Abfragefeld automatisch eine Abfrage nach folgendem Schema eingetragen:

```
SELECT * FROM Tabelle;
```

Im Bereich *Syntax* darunter können Sie auf Schlüsselwörter, Operatoren und Funktionen doppelklicken, um sich deren Beschreibung im integrierten MySQL-Handbuch anzeigen zu lassen.

Abbildung C-2 zeigt den MySQL Query Browser bei der Ausführung einer Auswahlabfrage, die alle Datensätze der Tabelle *rb_airlines* ausgibt. In der Tabellenansicht können Sie die Datensätze durch Klicken auf einen Spaltentitel einfach nach dem jeweiligen Kriterium sortieren lassen.

Längere Abfragen lassen sich als *Script* eingeben. Wählen Sie dazu zunächst *File → New Script Tab*. In dem großen Editorfeld können Sie nun eine beliebige Abfolge von SQL-Anweisungen eingeben. Mithilfe der Schaltfläche *Save* können Sie das Ergebnis als Datei speichern. Um die Abfrage später erneut auszuführen, kann sie mit *Load* wieder geladen werden.

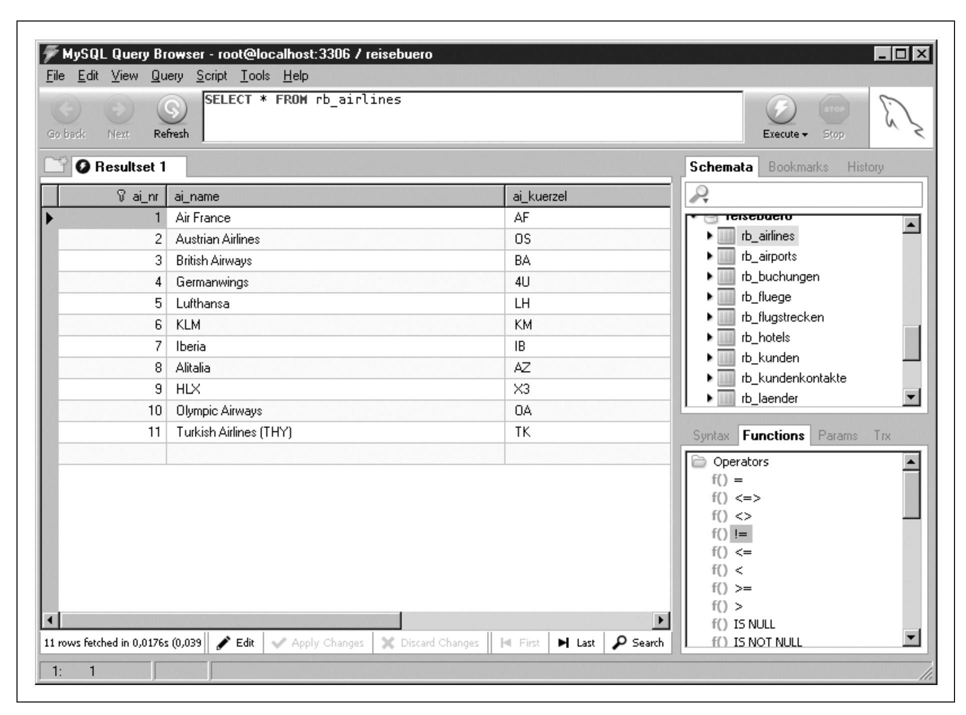


Abbildung C-2: Durchführung einer Auswahlabfrage im MySQL Query Browser

In diesem Anhang:

- Bücher
- Webressourcen

ANHANG D

Ressourcen und Tools

Einige Bücher soll man schmecken, andere verschlucken
und einige wenige kauen und verdauen.

Francis Bacon

Dieser Anhang gibt einen Überblick über weitere Ressourcen: Bücher und Websites, mit deren Hilfe Sie Ihre Kenntnisse nach dem Erlernen der MySQL-Grundlagen vertiefen können, sowie nützliche Zusatzsoftware.

Bücher

MySQL 5 von Michael Kofler. München 2005, Addison-Wesley. Ein umfassendes Handbuch zur neuesten MySQL-Version.

MySQL Cookbook, 2. Auflage von Paul DuBois. Sebastopol 2006, O'Reilly Media. In diesem Buch finden Sie unzählige praktische Anleitungen zur Arbeit mit MySQL-Datenbanken im bewährten O'Reilly-Kochbuch-Stil: Jedes »Rezept« setzt sich aus einer praxisorientierten Problemstellung, einem Lösungsansatz und ergänzenden Erläuterungen zusammen.

phpMyAdmin von Marc Delisl. München 2005, Addison-Wesley. Das offizielle phpMyAdmin-Handbuch in deutscher Übersetzung.

Webdatenbank-Applikationen mit PHP und MySQL, 2. Auflage von Hugh E. Williams, David Lane. Köln 2004, O'Reilly Verlag. Dieses Buch behandelt denselben Themenkreis wie das vorliegende Buch, ist allerdings eher ein Handbuch als eine Einführung. Daher eignet es sich gut als Ergänzung für Fortgeschrittene.

Apache 2, 2. Auflage von Sascha Kersken. Bonn 2005, Galileo Computing. Umfassendes Handbuch zum Apache-Webserver in den Versionen 2.0 und 2.2. Alle Installationsoptionen, Konfigurationsanweisungen, Module und Programmierschnittstellen werden umfassend beschrieben.

PHP 5 – Ein praktischer Einstieg von Ulrich Günther. Köln 2004, O'Reilly Verlag. Dieses Buch bietet eine gründliche Einführung in die Webprogrammierung mit PHP. Es erscheint in derselben Reihe wie der vorliegende Band.

Programmieren mit PHP, 2. Auflage von Rasmus Lerdorf, Kevin Tatroe, Peter MacIntyre. Köln 2006, O'Reilly Verlag. Der PHP-Erfinder Lerdorf und seine Koautoren stellen hier ausführlich alle wichtigen Aspekte der PHP-Programmierung vor.

Webressourcen

Die nachfolgende kleine Auswahl von Websites bietet weitere nützliche Informationen und zusätzliche Tools zu den verschiedenen Themen dieses Buchs. Da Webadressen oft schnell veralten, wird diese Liste auf der Website zum Buch jeweils aktualisiert und erweitert; die Adresse der Linkseite lautet: *http://buecher.lingoworld.de/mysql/links.php*.

MySQL, PHP und Apache

http://www.mysql.com/

Website der MySQL AB, der Firma der MySQL-Entwickler. Hier können Sie jeweils die neueste MySQL-Version herunterladen und erhalten weitere Informationen.

http://www.mysql.de/

Die deutschsprachige Website der MySQL AB.

http://dev.mysql.com/doc/mysql/de/

Die deutschsprachige Onlinedokumentation des MySQL-Datenbankservers.

http://dev.mysql.com/doc/mysql/en/

Die englische Version der Dokumentation. Für alle, die dieser Sprache mächtig sind, ist sie zu bevorzugen, da sie jeweils aktueller ist als die Übersetzungen.

http://www.phpmyadmin.net/

Homepage des phpMyAdmin-Projekts mit verschiedenen Informationen, der Onlinedokumentation und anderen Ressourcen.

http://www.php.net/

Website des PHP-Projekts. Hier können Sie die verschiedenen PHP-Versionen herunterladen, die Dokumentation lesen oder herunterladen und finden zudem Kontakt zu PHP-Usergruppen in aller Welt.

http://httpd.apache.org/

Die Site des Apache HTTP Servers. Wie üblich werden auch hier Downloads, Dokumentation und weitere Informationen angeboten.

http://www.apachefriends.org/

Apache Friends bietet mit XAMPP ein leicht zu installierendes Open Source-Paket, in dem eine (fast) fertig konfigurierte LAMP- beziehungsweise WAMP-Umgebung und weitere Servertools enthalten sind.

Zusätzliche Tools, Tutorials und Informationen

http://www.thekompany.com/products/rekall/

Rekall ist ein KDE-basierter Universal-Datenbankclient. Die Bedienung ähnelt Microsoft Access, ist also recht komfortabel. Rekall unterstützt MySQL als eine von mehreren zugrunde liegenden Datenbanken.

http://www.typo3.net/

TYPO3 ist ein beliebtes, sehr leistungsfähiges Open Source-Content-Management-System auf der Basis von PHP und einer Datenbank (normalerweise MySQL).

http://www.papaya-cms.com/

Auch dieses Open Source-Content-Management-System (an dem der Autor dieses Buchs mitarbeitet) ist in PHP geschrieben. Die Standarddatenbank ist ebenfalls MySQL. Papaya CMS setzt vollständig auf offene Standards – für Templates kommt etwa die offizielle XML-Transformationssprache XSLT zum Einsatz.

http://www.phpbb.com/

phpBB ist eine beliebte, PHP- und MySQL-basierte Open Source-Software zum Aufsetzen frei konfigurierbarer Diskussionsforen. Zahlreiche beliebte Foren-Communities im Web verwenden diese Software als Grundlage.

http://web.f4.fhtw-berlin.de/morcinek/sqltutor/

Leicht verständliches und übersichtliches SQL-Tutorial von Peter Morcinek, FHTW Berlin.

http://forums.mysql.com/

Diskussionsforen zu Themen rund um MySQL, in denen die MySQL-Entwickler mitunter selbst teilnehmen.

http://lists.mysql.com/

Übersichtsseite zu den offiziellen Mailinglisten der MySQL AB für Anwender, Entwickler und alle anderen. Nach dem Abonnement erhalten Sie alle Postings per E-Mail und können ebenso antworten oder neue Beiträge schreiben. Für die Teilnahme an Mailinglisten sollte ein automatisch sortierender Mailclient verwendet werden, da das Mailaufkommen recht hoch werden kann.

http://www.phpforum.de/

Eine deutschsprachige PHP-Foren-Community, in der über PHP, MySQL und verwandte Themen diskutiert wird.

http://www.apachefriends.org/f/

In den Foren der Apache Friends geht es neben Apache und dem Projekt XAMPP ebenfalls um MySQL und PHP.

http://www.4t2.com/mysql/

Hier können Sie eine (unabhängige) deutschsprachige Mailingliste zu MySQL abonnieren.

http://buecher.lingoworld.de/apache2/dirs.php

Erläuterung zahlreicher Apache-Konfigurationsdirektiven; die jeweils neueste lässt sich zudem als täglicher Newsletter abonnieren. Ein kostenloser Zusatzservice zu meinem Buch *Apache 2*.

http://dev.mysql.com/doc/refman/5.0/en/spatial-extensions.html

Die offizielle Dokumentation zur Verwendung von Geodaten in MySQL.

http://dev.mysql.com/doc/refman/5.0/en/error-messages-server.html

Vollständige Liste der MySQL-Fehlernummern und ihrer SQLState-Entsprechungen, jeweils mit Erläuterungen.

Index

B

C

D

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Über den Autor

Sascha Kersken kam 1983 zum ersten Mal mit einem Computer in Berührung und hatte später das Glück, dieses langjährige Hobby zu seinem Beruf machen zu können. Er arbeitet als PHP-Entwickler bei der papaya Software GmbH in Köln an dem gleichnamigen Open Source-CMS. Außerdem ist er Fachbuchautor, Dozent und IT-Berater mit den Schwerpunkten Unix-Serveranwendungen und Webentwicklung. Er ist Autor vieler Bücher, darunter *Praxiswissen Ruby, Praxiswissen Flash 8* (O'Reilly Verlag), *Handbuch für Fachinformatiker* und *Apache 2* (Galileo Press). Für den O'Reilly Verlag hat er unter anderem die Titel *Praxiswissen Dreamweaver 8*, *Ajax von Kopf bis Fuß*, *DNS & BIND Kochbuch* und *Active Directory* übersetzt bzw. mitübersetzt sowie das Buch *Apache – kurz & gut* aktualisiert und erweitert. Seine Freizeit verbringt er am liebsten mit seiner Frau und seinem Sohn oder mit guten Büchern.

Kolophon

Das Design der Reihe *O'Reillys Basics* wurde von Hanna Dyer und Michael Oreal entworfen, das Coverlayout dieses Buchs hat Michael Oreal gestaltet. Als Textschrift verwenden wir die Linotype Birka, die Überschriftenschrift ist die Adobe Myriad Condensed, und die Nichtproportionalschrift für Codes ist LucasFont's TheSansMono Condensed.